UNCLASSIFIED

AD NUMBER
ADB017872
LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to U.S. Gov't. agencies only; Test and Evaluation; 11 MAR 1977. Other requests shall be referred to Navy Fleet Material Support Office, Mechanicburg, PA.

AUTHORITY

FMSO ltr 7 Jul 1977

THIS PAGE IS UNCLASSIFIED

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*	2a. REPORT SECURITY CLASSIFICATION
Navy Fleet Material Support Office Code 9642 Mechanicsburg, PA 17055	Unclassified
	2b. GROUP

3. REPORT TITLE

Simulation Specification for modeling the Logistics Data Communications Network.

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

None

5. AUTHOR(S) *(First name, middle initial, last name)*

Network Analysis Corporation
410 Pine Street
Vienna, VA 22180

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
Oct 976	79	None

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00104-76-D-5532	
b. PROJECT NO.	NAC / FR-095 02
	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
c.	None

10. DISTRIBUTION STATEMENT

Limited distribution - U.S. Agencies only

11. SUPPLEMENTARY NOTES

None

12. SPONSORING MILITARY ACTIVITY
Naval Supply Systems Command Washington, DC 20376

13. ABSTRACT

A simulation strategy that can be used to model the Logistics Data Communications Network (LDCN) is described in detail. The various components of LDCN and their interconnections are set forth. The modeling strategy is reported with particulars including the inputs and outputs of each model in the system. Also presented is a scenario life-cycle for a transaction from the origination at the terminal through the network and back to the terminal.

KEYWORDS
  simulation
  modeling strategy
  simulation program structure
  LDCN global network model
  LDCN host model
  LDCN Front End Processor (FEP) model
  LDCN concentrator model

SIMULATION SPECIFICATION

FOR

MODELING THE LOGISTICS DATA

COMMUNICATIONS NETWORK

October 1976

Prepared For

Fleet Material Support Office

Naval Supply Systems

Mechanicsburg, PA 17055

Code 9642

FR. 095.02

Prepared by

Network Analysis Corporation

410 Pine Street

Vienna, VA 22180

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The Navy Fleet Material Support Office (FMSO) is in the implementation phase of an integrated real-time data communications network to make logistics information available to a large number of dispersed Navy users. This information is necessary for effective and efficient logistics support. The Logistics Data Communications Network (LDCN) will provide direct access from remote locations to two Inventory Control Points (ICPs) which maintain and manage major logistic data bases. Each ICP computer complex has two U494 computers; one also has an IBM 370 while the other has a Burroughs 3500. Each complex will have a Front-End Processor (FEP) to provide the communications functions needed and relieve the Host computers of these tasks. Programmable communications concentrators at selected Navy sites will be used to interface low-speed terminals and provide cost-effective remote access. Each concentrator will be connected via high-speed full-duplex lines to each of the FEPs.

To assist management of the operational system and future developments to satisfy new user requirements, FMSO has identified the need of an effective simulation tool to model system behavior under various load conditions and design variations. Such a tool can be used to assist operations personnel when unresolved problems occur in the existing network. The results of a simulation run can be used for detailed examination of each system resource utilization and to trace the progress of activity in the system, thereby giving insight to a problem that empirical observation cannot. In addition, a simulation program is essential when expanding the capabilities of the network. It can be used to isolate potential bottlenecks and quickly and cost-effectively evaluate potential alternatives to alleviate such bottlenecks before the capabilities are required and implemented.

When modeling a single processor, as opposed to the entire network, simulation provides an economical means of evaluating various design alternatives, both hardware and software. The primary objective of processor modeling is to economically attain designs satisfying the performance requirements.

However, the effect of various performance requirements on a particular processor configuration can also be obtained via simulation.

In general when dealing with such complex systems, simulation can provide management and operations personnel with the off-line examination of system operations needed for future growth feasibility studies as well as a tool for verification of the actual implemented system.

This report describes a simulation strategy that can be used to model the LDCN. The strategy presented allows total system modeling (i.e., the network) and system element modeling (i.e., the processors). We begin in Chapter 2 by giving a description of the various components of LDCN and their interconnections. We also present a scenario life-cycle for a transaction from the origination at the terminal through the network and back to the terminal. In Chapter 3 we present the modeling strategy. In Chapter 4 we describe in detail each of the models presented in the previous chapter. The inputs and outputs of each model are described in Chapters 5 and 6 respectively. In Chapter 7 a methodology for linkage of the various models is discussed. Implementation considerations and, in particular, how various simulation languages could be used to handle the specialized problems faced in coding the programs are described in Chapter 8. Finally in Chapter 9 the conclusions and recommendations are presented.

2. SYSTEM DESCRIPTION

The Logistics Data Communications Network (LDCN) is composed of five classes of devices:

1) Host Computers or simply Hosts

2) Front-End Processors - FEP

3) Concentrators

4) Terminals (e.g., CRTs, RJE, TTY)

5) Communication Lines

In terms of the simulation strategy we will exclude the terminals from consideration as they are not programmable and can be viewed as sources and sinks of traffic. It is essential, however, to include the low-speed lines that connect the terminals to the concentrators.

LDCN is composed of two main computer complexes; one ICP at Mechanics-burg, Pennsylvania (SPCC), and one ICP at Philadelphia, Pennsylvania (ASO). At Mechanicsburg, the ICP computer complex contains two U494 computers and one IBM 370. At Philadelphia, the ICP computer complex contains two U494 computers plus a Burrought 3500. At each ICP, one U494 supports real-time processing activity for an Inventory Control (IC) data base, and the other U494 supports real-time processing for a Weapons System (WS) data base. Each U494 can access the other's data base in a read-only mode. The 370 and B3500 also support real-time processing but have access only to their own data base.

In addition, each complex has a Front-End Processor (FEP) minicomputer to serve the communications functions for the host computers. The FEP is a dual-CPU Interdata Model 7/32 minicomputer with moving and fixed-head disks and several tape drives. The FEP is connected to each Host via a half-duplex hard-wired channel.

A total of nine remote concentrators are planned for the system; each concentrator connected to both FEPs. The concentrator is an Interdata 7/32 minicomputer with limited peripherals. The FEP-to-concentrator connections will be via high-speed dedicated full-duplex channels.

A variety of terminals will be accessing the concentrators via low to medium speed half-duplex dedicated or dialup circuits. The configuration of LDCN is shown in Figure 2.1.

The system is transaction oriented with the terminals entering transactions via the concentrators, which are then forwarded to the FEP where some fraction are in turn forwarded immediately to a Host, and the others are spooled for batch processing. Outputs generated by the Host are immediately transferred to the FEP where they are buffered for segmented delivery to the concentrator followed by transmission to the originating terminal. The life cycle of a transaction is the sequence of steps describing its progression through the system from its origin to its termination. Examining the life-cycle of a typical transaction will provide substantial insight to the various simulation models and the rationale of their structure. Transactions originate at the terminal where a logged-on user keys in the message. The characters are buffered in the concentrator where editing by the user can be performed. After the end-of-message signal is sent, the concentrator transmits the transaction to the FEP via the high speed lines. The FEP will be a dual-CPU configuration; each CPU will effectively be connected to the Hosts and the concentrators. Each CPU can take messages from the concentrators and deliver them to the appropriate Host and vice versa. In addition, one CPU can receive a message from either Host or concentrator and the other CPU can deliver it. When one CPU fails, the FEP will continue to function. As transactions are received by the FEP, they are processed for error detection and to verify user validity, program availability, and type. Two basic transaction types are identified: Real Time Transactions (RTT) - those to be sent immediately to a Host on a First-In/First-Out basis, and Possible Batched Transactions (PBT) - those to be spooled on disk until a batch criteria is satisfied (time or volume) and then transferred to a Host.

2.3
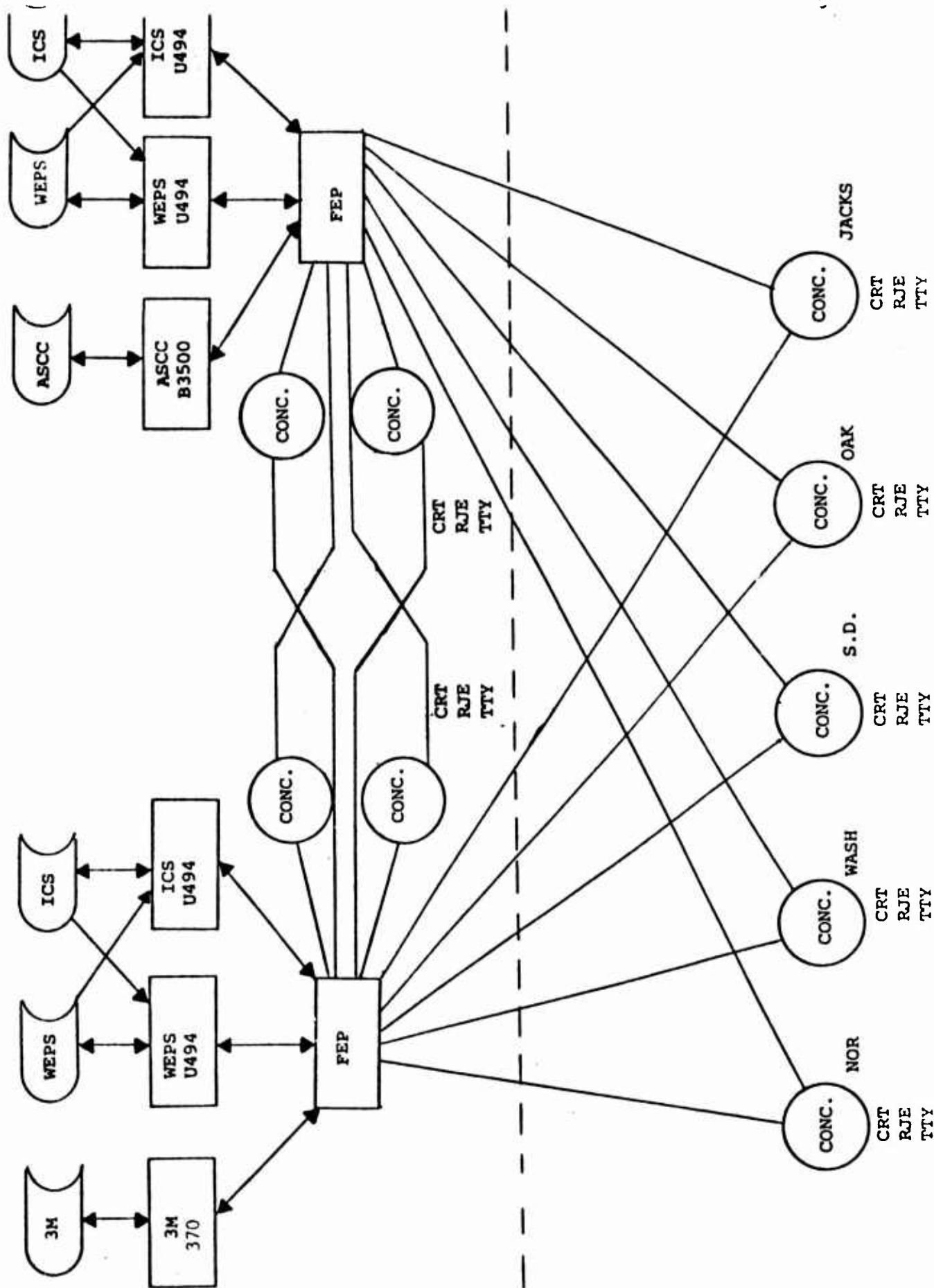
Two types of Real-Time Transactions for the U494's are also identified:

1) Upquires - those RTT's that can be sent only to a specific Host because they imply a write operation (update of data), and

2) Inquires or shared transactions - those RTT's that can be sent to either 494 because they imply a read-only operation. Both 494's has access to both WEPS and ICS data bases in a read-only mode.

In the FEP there are 2 queues (each FIFO) for each 494 Host; one for the upquiry real-time transactions, the other for the batch transactions. However, there is one additional queue for the 2-494's that hold the shared real-time transaction. (The decision as to which 494 gets these shared requests is done by an algorithm in the FEP that receives utilization information from the 494's in the header of received transactions.) The procedure for messages destined for one of the 494's is to alternate between the 3 queues (real-time, shared real-time, and batch) and transmit the message (if any is present) at the top of each queue. For the other Hosts (370, B3500) there will only be 2 queues - one for real-time, the other batch. Both queues are FIFO.

Transaction processing by a Host results in an output to be returned to the user. This output is usually quite long, and since concentrators do not have auxiliary storage, the content must be buffered via auxiliary storage at the FEP. Thus, when outputs are received by the FEP, they are transferred to disk to queue for buffer availability at the concentrator for transmission to the user. Double buffering for each user communications facility is used at the concentrator to obtain continuous transmission of outputs to the user. As each buffer is emptied, transmission of the next buffer is initiated, and a request for a new output segment (or RFNM for Request for Next Message) for the emptied buffer is sent to the FEP. The FEP then reads the next output segment from disk and transmits it to the concentrator. The life-cycle of a transaction terminates after the last output block is transmitted to the terminal.

3. MODELING STRATEGY

Development, operation, and management of a network such as the LDCN requires the facilities to predict network and network subsystem performance as a function of traffic characteristics and design variations. During the initial development phases, such a capability can be used to determine appropriate sizing factors and configurations for the given performance objectives and traffic requirements. As development proceeds, it can be used to examine sensitivities to changes in requirements and to determine appropriate corresponding configuration adjustments. Once the network is operational, such a capability can be used to assess the significance of trends in traffic changes and to predict necessary design adjustments before the need becomes critical.

The capabilities described above are basic objectives for development and use of a system simulation model. The LDCN is a complex system, composed of major subsystems in concentrators, Front End Processors (FEPs), and Hosts, unified via data communications lines into an operational system. Major design issues can focus on the network as a whole, or on any subsystem, and may deal with hardware or software. Development of a simulation model as a general tool to enable examination of such issues requires major consideration be given first to the overall modeling strategy. The strategy must address the issues of modeling the individual subsystems as well as modeling their interaction in a unified system. The purpose of this report is to present an appropriate strategy for modeling the LDCN.

The disucssion presented below outlines a modeling strategy that allows both types of results. We begin with the global network strategy and then show how detailed processor models can be extracted from the global model.

3.1 Global Modeling Strategy

As an input to development of a total system modeling strategy, the entire network of Hosts, FEPs, concentrators, and communications lines has been functionally defined and portrayed in a simple system schematic.

3.1

Previous work has defined the basic operational, hardware, and software characteristics envisioned for the implemented system, and the traffic load and flows it is required to support. With these factors given, two basic simulation strategies for global network modeling may be considered:

1) Detailed Simulation;

2) Simplified Simulation But With Selected Detailed Subsystems.

The first choice is to make one large program that encompasses the entire network. Each processor would be modeled at the desired level of detail. From such a program would come results on the inner workings of each processor as well as global network performance measures. This concept is obviously infeasible for the LDCN due to a variety of reasons:

1) The number of computers involved in the network (in LDCN –
   6 Hosts, 2 FEPs, 9 concentrators) would make the program
   exceptionally large and require extensive computing resources
   for execution;

2) The program would be required to simulate, in detail, the same
   type of device several times concurrently. This is certainly
   wasteful. One should simulate the same processor only once
   for a given set of conditions;

3) There exists a time unit disparity. The level of detail desired
   for the processors would be in the micro to milli-second domain
   whereas the communication line delays are on the order of a milli-
   second to a second. All the simulation time would be spent in
   the devices, processing the message. Very few messages would be
   transmitted between the processors. The execution time of the
   simulation model may be an order of magnitude greater than the
   elapsed time of the actual network being simulated.

The second basic strategy for simulating the network as a whole would be to simplify and reduce the most detailed portions of a model similar in concept to the one presented above. The most detailed portions of such a model are found in the processors. Their basic functions as communications processors, are to receive messages, apply various validity and error-correcting functions to the messages, and then to forward the message onto the next processor. This is a much simplified description of their functions but it is, in essence, their primary task. When evaluating the total network performance, the designer is not interested in the specifics on how the processor performs its task. He would simply like to know that given an input message, the processor will generate some form of an output at a finite time later. If this delay can be quantified and replaced by an aggregate expression, which we will call a service time distribution, the problem of modeling the entire network becomes tractable. The network can be represented as a flow chart of single-servers with queues and storages in the appropriate places (see Figure 3.1). A simulation program depicting this model becomes straight forward.

3.2 Simplified Simulation With Selected Detailed Subsystems

The above discussion may appear to over-simplify the task of global network modeling, but simplified simulation with selected detailed subsystems has several advantages that make this approach attractive:

1) The model and subsequent simulation program are cost-effective; the program would be relatively simple to code. The execution of such a program would be efficient thereby allowing many design experiments to be run with each run simulating a longer real-time period.

2) The results of interest from such experiments should and would be the global performance statistics only. Time and money are not spent in obtaining detailed statistics associated with a particular processor. These results will be determined by selectively incorporating detailed subsystem models in the experiments.

Host

Half Duplex Channel

Front End

Full Duplex High-Speed Lines

Concentrator

Half Duplex Low-Speed Lines

Traffic
Generator
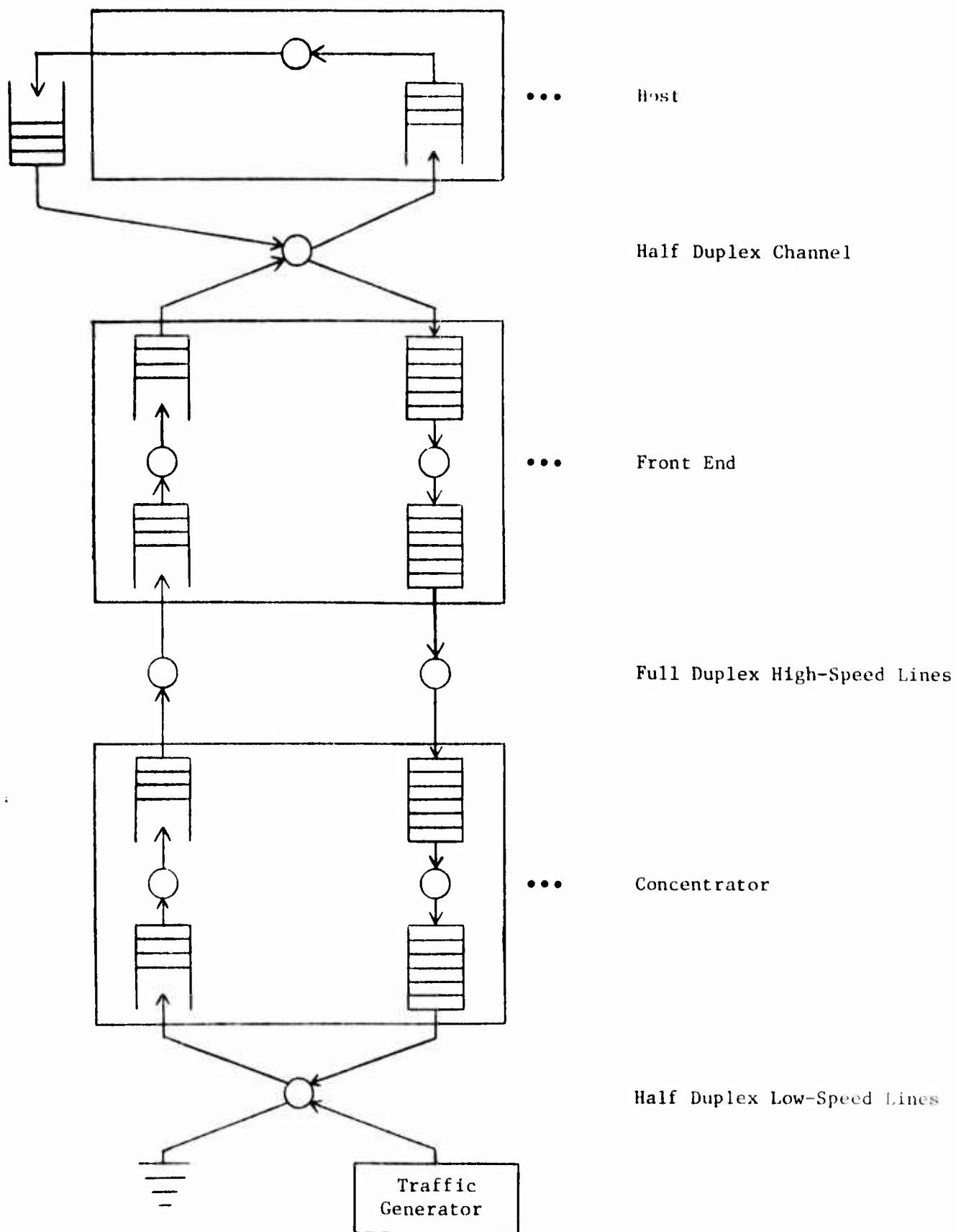
Figure 3.1

NETWORK FLOW CHART

3) By effective program representation of each device, such as by a macro in GPSS or a matrix entry in FORTRAN, reconfiguration of the network, either by adding or removing devices, requires minimal, if no, program modification.

The service time distribution can initially be obtained by best estimates of what the service time will be in each processor at zero-load conditions. In some cases, for example the Host, the service time might already be known.

Using this service time as a constant, independent of load conditions, preliminary experiments can be quickly undertaken. One set of experiments could be the evaluation of the performance of the network under varying traffic loads and patterns. A base-line traffic pattern could be obtained from existing statistics and future projections. The sensitivity of varying a class of processors' service time can also be examined. Early detection of potential bottlenecks can take place. In addition, initial design constraints can be formulated.

The above discussion outlines a simulation strategy that can be used and implemented for quick and easy evaluation of a communications network such as LDCN. However, the results of such an evaluation are clearly suspect due to the many assumptions being made to simplify the model and reduce unknowns. Inappropriate assumptions might produce results that eventually turn out to be orders of magnitude wrong. This global model is meant to provide the first gross performance trends quickly and cheaply and to act as the framework for more detailed models as it evolves into an accurate global performance measurement tool. The weakest element of this model is the fact that the service time of the processors is obtained by estimation. This can be remedied by the selective incorporation of detailed subsystem models without incurring the negative effects of a detailed model for the total system.

3.3 Selective Detailed Models

We mentioned above that one goal of the simulation effort is to acquire
the ability to model, in detail, particular processors. This can easily
be accomplished for stand-alone machines. Simulation of computer systems
has evolved out of the evaluation of a single processor, usually a main
computer. Typical studies are concerned with the effect of operating sys-
tems, memory allocation, peripherals, etc. The problem we are faced with
here is when the processor is an element of a total system network.

Typically the major areas of concern are the sequence of events that
occur inside the device; i.e., what happens to the quantum of interest (job,
task, transaction, message, etc.) after it enters the device and before it
leaves. What transpires before it enters and after it leaves is usually
not important.

But communications processors are bi-directional devices. Typically
after a message passes through the processor on the upstream path, it or
some function of the original message reappears at the processor traveling
in the downstream direction. The simulation model must, of course, account
for this bi-direction flow of traffic.

The interaction between the modeled processor and its surrounding
environment is an integral part of the simulation. The protocols, imple-
mented and envisioned, dictate a majority of the functions required at a
processor. Since there is this heavy interaction in two directions, and
the fact that once a message leaves a processor, it is usually not terminated
but reappears, the external environment of a communications processor is
much more important that in a stand-alone machine. In fact, the exogenous
events are the ultimate driver of the model itself.

How does one correctly choose the external environment for the com-
munications model? This is where the global network model comes into play.
One simplified single-server model of the particular device in the global
network is replaced by the more detailed version of the model. Next, since

3.6

most devices do not communicate with devices of the same type (as in LDCN), all other single-server models of the same class in the global model are removed. Further reductions are made until what remains is one detailed model of a particular processor with the external environment being all the simple·single-server modeling devices that directly communicate with it. This scheme will become clearer in the next chapter when we apply this technique to LDCN. But the aim is to maintain as much of the global network model as possible when performing detailed modeling of a processor.

However, the simulation program that arises from this type of model might still be too large and complex. In this case, reductions in the structure of the external environment should occur first. There might be a point, however, where further reduction of the environment could lead to inaccuracies and inflexability. If the program is still too large, minor features of the detailed model might be removed. This would be least attractive, but it should be kept in mind that the external environment of the detailed model is almost as important as the model itself. Yet the purpose of the detailed model is to perform detailed modeling. What to sacrifice is a critical decision to make and can only be made at implementation time.

3.4 Simplified Detailed Model Linkage

Once the detailed model is built, accurate descriptions of a device's service time distribution can be made. These distributions can then be used in the global network model, replacing old approximations. Many service time distributions can be obtained for various hardware (peripheral and memory) configurations at the device. The global network program can then be run using the various distributions to obtain the global effect of such configurations. The distributions will also serve, eventually, as the more accurate description of the external environment of some other processor.

3.5 Basic Modeling Strategy

The problem of modeling and simulating a complex data communications network, such as LDCN, can be approached with the strategy proposed above. This strategy can be used to satisfy the two basic goals of such a simulation effort:

1) The ability to obtain simulation results for the entire network;

2) The ability to obtain simulation results for any particular processor or device in the network.

The strategy is a two-fold hierarchical plan. The first milestone and the highest level of the hierarchy is the development of a gross simplified queueing model of the entire network. Each processor is modeled as a single server. Service times are obtained from best estimates, design goals, or existing statistics. Once the global network model is formulated, the development of each of the detailed processor models can be initiated. Meanwhile, the global model can be coded and experiments quickly started. The development of each of the detailed models can take place in parallel or sequentially, as no one model is directly dependent on another. The structure of the detailed model is obtained from the framework of the global model as described above. The external environment of any detailed model is critical and should not be overlooked. The detailed model can be used to simulate the processor at any desired level. If program size becomes critical, two alternatives exist:

1) Reduce the scope of the model and therefore hopefully the size;

2) Replace a complicated section of the program by a simpler one and model this portion in detail in a stand-alone mode. This concept can be viewed as another level of the hierarchy of the models.

As results are obtained from the detailed model, they are fed back into the global network model and are used in modeling the external environment of other detailed models.

As the global model is fine-tuned with results from the detailed models, more confidence can be placed in the global performance results. More meaningful experiments can be undertaken. One such series of experiments could be the evaluation of future enhanced capabilities. The possible scenario for one experiment could be the following:

1) Use the global network program with service time distributions for the processors that were obtained from the most current configuration, traffic load and pattern;

2) Run the program, introducing the new set of conditions, e.g., increased traffic load;

3) Isolate any bottlenecks that occur in the model, e.g., a saturated processor;

4) Run the detailed model of the saturated processor using the same set of new conditions;

5) Isolate the cause of the bottleneck locally;

6) Reconfigure the processor until the bottleneck is removed. This might require that the detailed model be run several times;

7) Using the new service time distribution obtained from 6), return to the global network model and rerun the experiment;

8) If performance requirements are met, then the new capability can be accommodated and the steps needed to provide it are known.

9) Otherwise isolate the new bottleneck (might be the same one) and go to that processor's detailed model;

10) Continue with Step 6.

Given this hierarchical approach to network modeling, we will see, in the next chapter, how it can be applied to the specific case of LDCN.

4. SIMULATION PROGRAM STRUCTURE

The common structure of an integrated model is a modular program
functionally divided into three main areas:

1) Monitor;

2) Utilities;

3) Models.

The organization has many parallels to common operating systems with execu-
tives, utilities, and application programs. The monitor is the coordinating
element for all the modules and real-time activity. The utilities are pro-
gram modules handling routine functions that are independent of a particular
model implementation such as report generating. The models are the basic
program modules. Each module models some particular component or function
of the system. In this chapter each area of the common structure is
briefly described; the models being described in more detail.

4.1 Monitor

The level-one structure of the modular program is portrayed in Figure
4.1. Each of the main functional areas interacts with the other two and
with a common data structure maintaining facility tables, queues, event
chains, state vectors, etc. The monitor serves to coordinate the flow of
events in the system. It has primary responsibility for maintaining the
endogeneous and exogeneous event chains. As events progress, it advances
the simulation clock and at appropriate times invokes the utility routines
necessary for system statistical snapshots, state-recording, etc. It moni-
tors operator input messages and exercises control over simulation resources.
In some language implementations, some form of the monitor is automatically
provided.
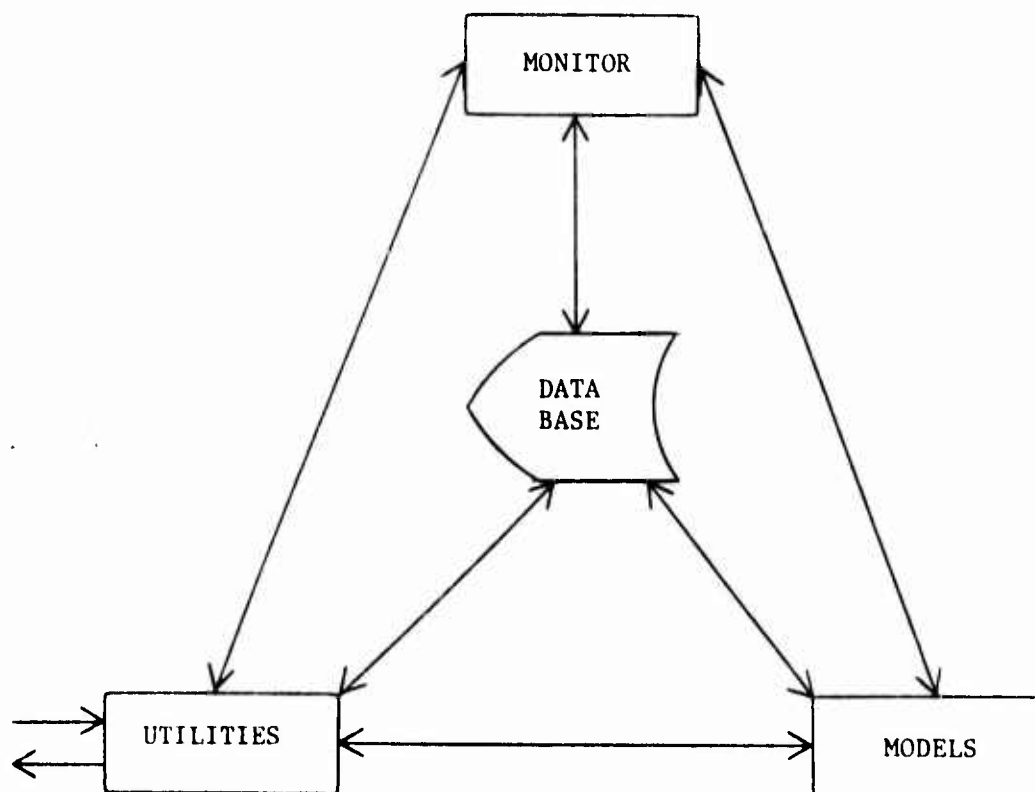
Figure 4.1 - COMMON PROGRAM STRUCTURE

4.2 Utilities

The utilities are divided into three main areas:

1) Front-End (I/0);

2) Traffic Generation;

3) Statistics

Each of these areas is further subdivided to yield the basic structure shown in Figure 4.2. An important part of the utilities package is its common interface structures with other program modules. All interfaces are through the three main area levels and are subject to coordination control of the monitor.

4.2.1 Front-End (I/0)

The front-end modules of the utilities provides both the inputs to the entire program from the user and the outputs from the program to the user. In terms of input, probably the most convenient and cost-effective mechanism is via an interactive program. Such a front-end could allow the entering of parameters, control options, etc., in free format and provide extensive prompting for and error detection of such input. This front end could create a batch-type file that contains the JCL needed to run the simulation program along with the required input data. This file then could be submitted via RJE to the simulation programs. If a graphics capability is available, the network can be displayed, quickly verifying its configuration.

The output of a simulation run is usually voluminous. Listings are the typical form of output. Report generators are the modules that perform this task. However, along with an interactive input module, the output could also be viewed with interactive programs. Specific reports and text could be scanned with text editors. If graphics are available, histograms,

1. Front End (I/O)
1.1 Interactive Interface
1.2 Input
1.3 Report Generator
2. Traffic Generator
2.1 Stochastic Source
2.2 Attribute Descriptors
3. Statistics
3.1 Statistics Gatherer
3.2 Trace
3.3 Statistics Processing

Figure 4.2 - UTILITIES STRUCTURE

Network Analysis Corporation

distribution and throughput-delay curves can be viewed quickly. It is even possible to superimpose several curves to immediately compare results. Figures 4.3 and 4.4 are examples of interactive graphic output displays.

4.2.2 Traffic Generator

Essential to every simulation program is the traffic generator. It serves as the driver of the models. Transactions are usually characterized as following a random distribution in time with the mean arrival rate being used as a measure of demand on the system. At generation time, attribute descriptors are also created. Attribute descriptors define values for attributes of a transaction that are also specified by random distributions. Such attributes may include input message length, input duration (for operator entry of a transaction on a character basis), output message length, priority, etc. The user should be able to specify such distributions through a general, flexible, all-purpose traffic generator that can be used by all simulation programs.

4.2.3 Statistics

The primary form of output of any simulation program is the statistical information describing system behavior. Three functions are identified in obtaining statistics. Modules are needed to gather the statistics as the simulation program is running. As the state of a facility changes or as a transaction leaves a certain processor, statistics are recorded. The ability to trace a particular transaction through the model not only provides the analyst with more information about system behavior but assists the programmer in the debugging phase of program development. After the simulation program is finished running, various post-processing modules are required to transform the raw statistical data into utilization, distribution, etc., results. These post processors can be part of the simulation program itself, taking the data from internal data structures, or they could exist in a stand-alone mode reading in a file that was dumped by the simulation program. In either case, they would have to interact with the report generator modules in the front-end portion of the program.
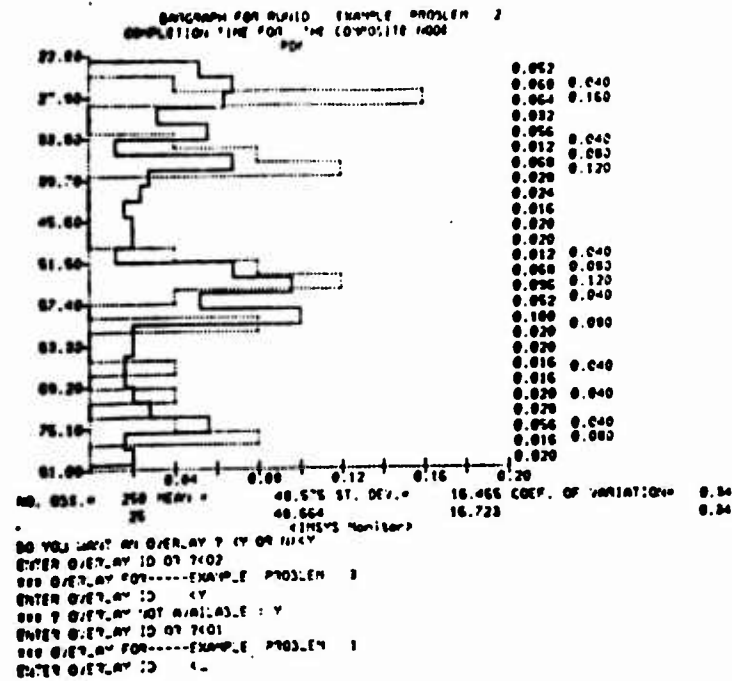
Figure 4.3

EXAMPLE OF INTERACTIVE GRAPHIC OUTPUT

ENTER OUTPUT COMMAND OR ? <
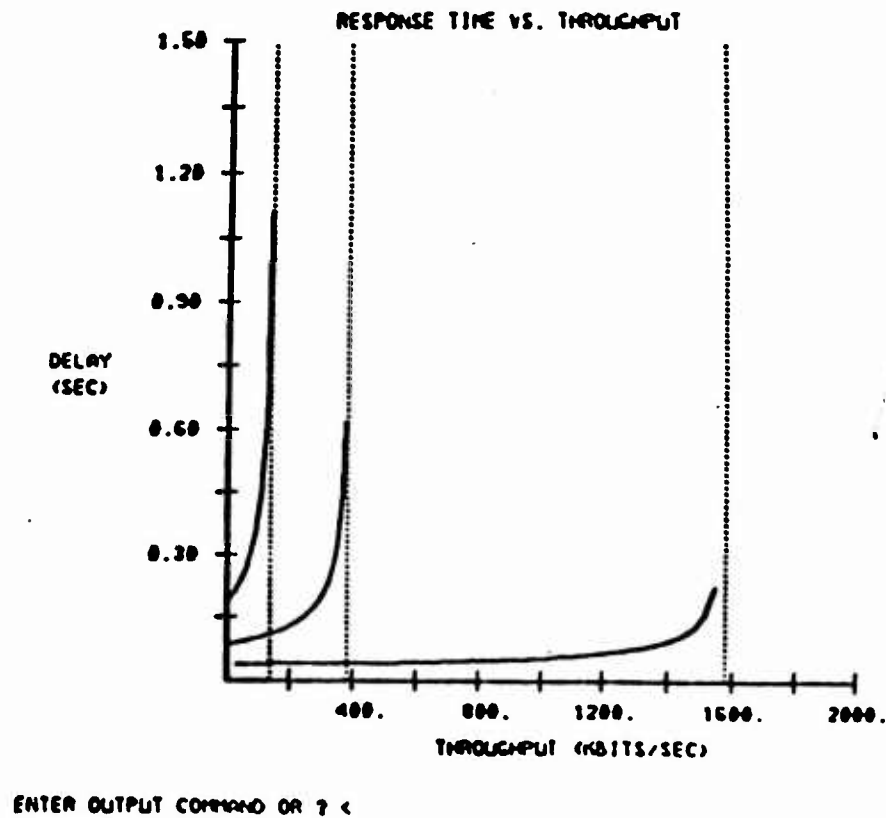
Figure 4.4

EXAMPLE OF INTERACTIVE GRAPHIC OUTPUT

4.3 Models

The third component of the simulation program is the model module. We now present the flow charts depicting the models for the LDCN devices. We begin with the global network model, describing the sequence of facilities visited by a typical transaction. We then describe each processor model (Host, FEP, conc.) in the context of the global network and in more detail as a stand-alone program.

4.3.1 The LDCN Global Network Model

The schematic diagram for the LDCN Global network model is shown in Figure 4.5. A couple of general comments are given before we proceed. First, the model represents the configuration and functions of LDCN to the best of our knowledge. If say, a storage facility is actually implemented as a queue, it is easy enough to modify the model to reflect this. Half-duplex lines or channels are modeled as a single server in both directions; full duplex lines are modeled as two servers in one direction each. It is also easy enough to modify the model when the line status changes from half duplex to full duplex or vice versa. The communications processors (Front-End Processor and Concentrator) are modeled as two servers; each server functioning in one direction, upstream or downstream. This allows the measurements and service times to be taken for both input and output which typically require different processing and therefore imply different delays. However, the concentrator is implemented as a single-CPU processor and even though the FEP is a dual-CPU configuration, each FEP-CPU is not dedicated exclusively to input or output. The division of functions in the model is meant only as a convenience in implementing different service times. The exact model should reflect that only one CPU or two CPUs "in parallel" are performing both functions.

We now describe the sequence of events, or life-cycle, that a transaction would take through the global network model.

1) The transaction is initiated by the traffic generator at the concentrator. We assume that the transaction is one complete
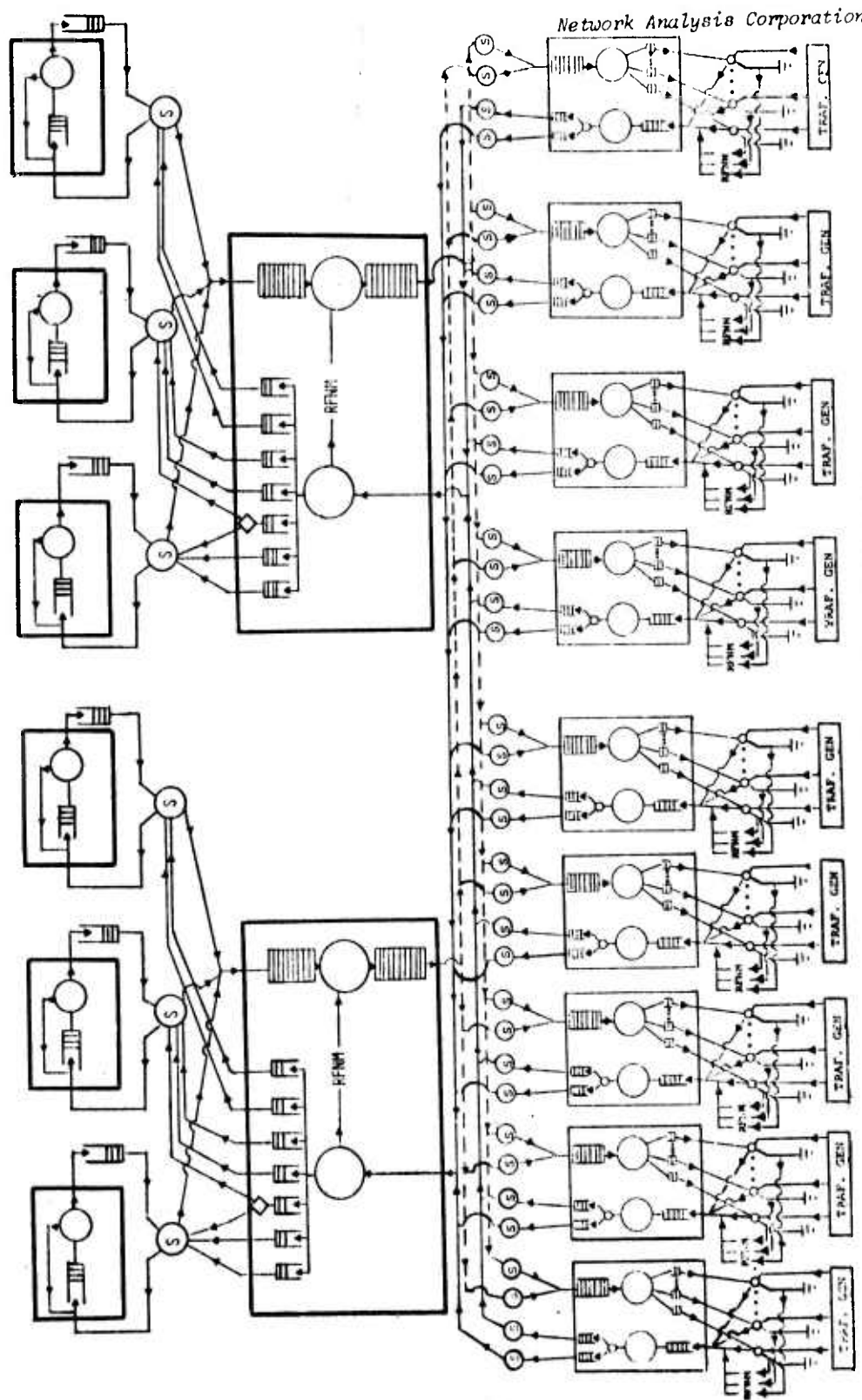
FIGURE 4.5: THE GLOBAL NETWORK MODEL

Network Analysis Corporation

4.8

message. Several options exist at the level of detail for the traffic generator. There could be one traffic generator per terminal, one per concentrator, or one for the entire system. We feel that the way we have depicted the traffic generator here for the global model (one per concentrator) allows for both general and specific characterizations of the traffic pattern and load. Consequently, the traffic generator must tag the transaction, in addition to other attributes, with the origination terminal i.d.

2) The transaction then seizes the low-speed line server; here represented as half duplex. The service time is known and is equal to the transmission rate of the line.

3) The transaction then queues for service at the concentrator.

4) It then seizes the upstream concentrator server. The service time is obtained initially from estimation but subsequent by results from the detailed concentrator model.

5) After being served, the transaction queues for output to the FEP.

6) After seizing the high-speed line server for a duration equal to the transmission rate of the line, it enters the FEP.

7) If the message was a RFNM (request for next message), it then progresses to the downstream FEP server. Otherwise, the upstream module processes the transaction in preparation for transmission to the Host. Again the service time is obtained either by estimation or detailed modeling results.

8) The transaction is then placed on an output queue depending on destination Host and transaction type.

9) After seizing the half-duplex channel server, for a duration again equal to the transmission rate, the transaction enters the Host.

4.9

10) The transaction queues for processing after which the Host server performs part or all the processing required. After completion of the processing, the transaction is now an output message which begins its downstream journey.

11) After queueing for the channel server and being served, the message enters the storage buffers in the FEP.

12) After being served by the downstream FEP server, output blocks are then sent to storage buffers awaiting transmission to the concentrator.

13) After seizing the high-speed line server the output block enters storage buffers at the concentrator.

14) Subsequent to being processed by the downstream concentrator server, two output blocks are prepared for transmission on a per terminal basis.

15) After seizing the half-duplex line server, each output block is terminated and a RFNM generated for transmission back to the FEP. This RFNM joins the upstream input queue at the concentrator.

As the processor models are presented below, each of the above steps relevant to each model will be described in more detail.

4.3.2 The LDCN Host Model

The Host model is depicted in Figures 4.6 and 4.7. Figure 4.6 shows the relationship between the Host model and the global network model. Figure 4.7 shows the Host model in a stand-alone mode. Notice that the only element in the external environment is a reduced version of the FEP model. In addition, the simple single-server model of the Host is now replaced with the more detailed version. This basic model can be used to

FIGURE 4.6: HOST MODEL

t_i t_o Service time of
Host $= t_o - t_i$

Half Duplex Channel

Front End
Output
Storage

to other 494*

UPQ   INQ

Batch

TRAFFIC
GENERATOR
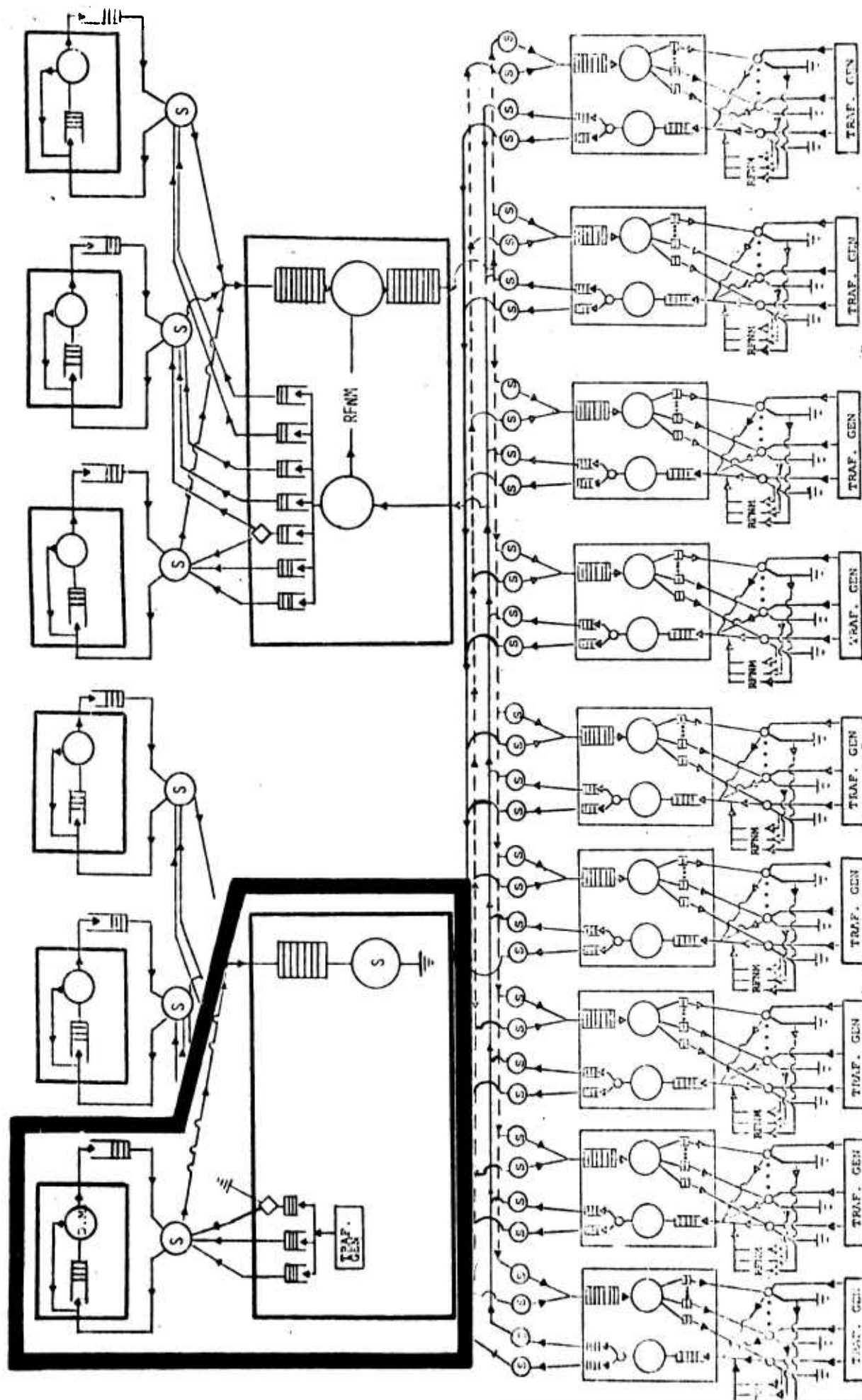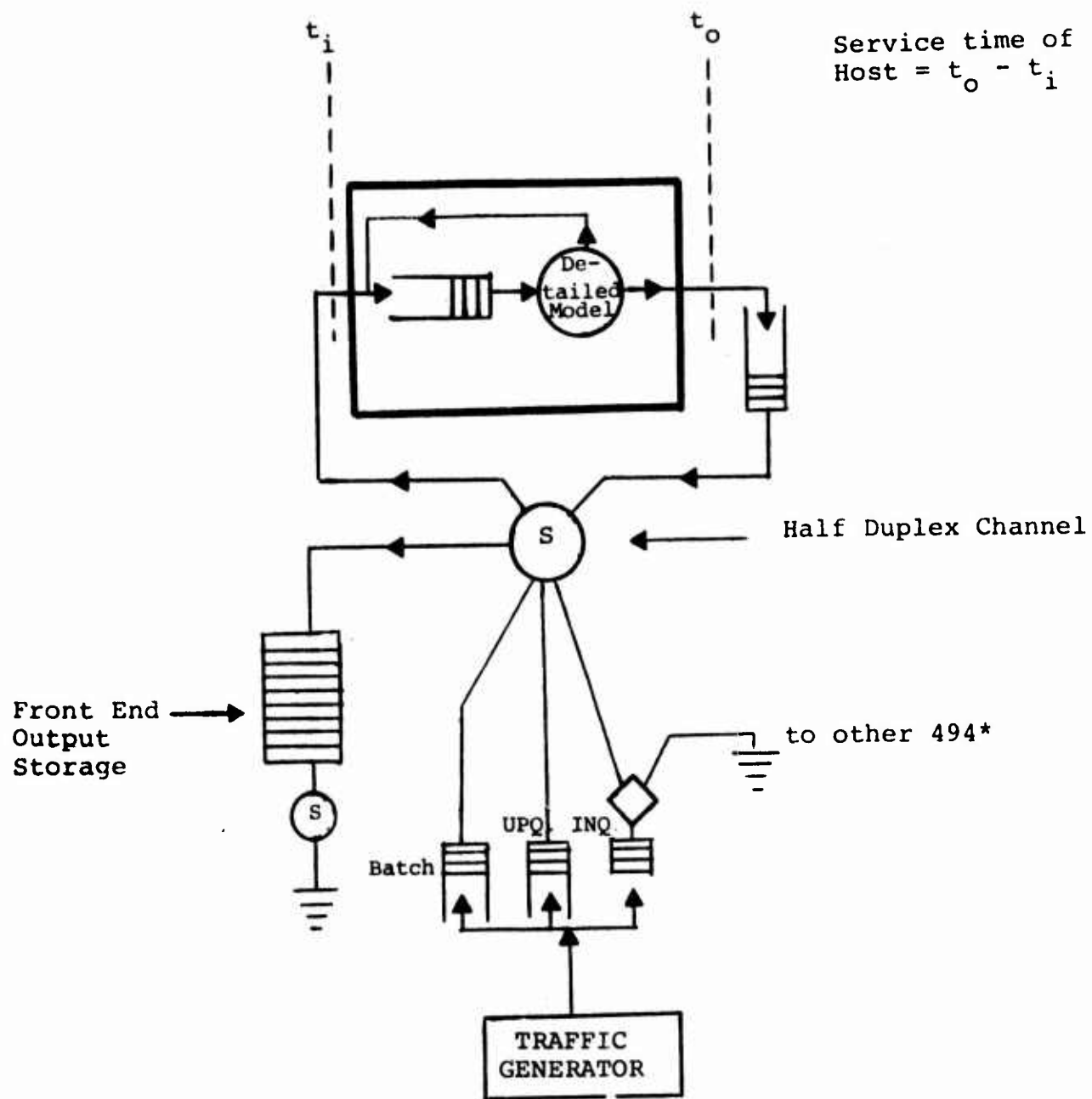
FIGURE 4.7:   494 - HOST MODEL

* If modeling a non-494 Host, remove Inquiry Storage and Path.

model any type of Host in LDCN. We will now trace the life cycle for
transactions in this model.

1) Transactions are generated and immediately placed on an output
queue at the FEP. There are two queues; one for real-time trans-
actions, the other for batch transactions. However, when modeling
a U494 Host, there is an additional facility; i.e., a storage,
that is used for real-time transactions that can be processed at
either 494 (inquiries). We will describe how this mechanism is
used to eliminate the necessity of modeling the two 494s simul-
taneously.

The traffic generated for the Host is only the amount to be pro-
cessed by the particular Host under consideration. However, when
modeling a U494 Host, the inquiry traffic generated should be the
amount destined for the complex under consideration (ASO or SPCC).
As an inquiry transaction is generated, it attempts to enter the
storage mechanism (really a finite-length queue) in the FEP. If
the storage is full, the transaction at the head of the storage is
removed and terminated, thereby making room for the new trans-
action which then joins the storage. Terminating the transaction
in the above manner simulates an inquiry transaction that even-
tually is processed by the other 494 Host. This mechanism
prevents the buildup of a "standing-wave" of inquiries queued for
a 494 when the 494 becomes momentarily unavailable during processing
of a large batch of transactions. It is during such an interval
that the load-sharing feature of having a second 494 should be of
greatest significance. Without such a feature, the "standing-
wave" effect would result in serious performance degradation on
inquiries for a period exceeding the momentary service disruption.
By varying the length or size of the storage, one is able to vary
the amount of inquiry transactions processed by the Host. A zero
length storage would imply no inquiry processing; an infinite
length would mean that this Host process all the inquiry trans-
actions. Measurements must be taken on the amount of inquiry
traffic terminated and it should be verified that the other U494
is able to accomodate at lease that amount. The scenario could be:

4.13

a) Run one 494 model with all the inquiry traffic destined for
    the system using a finite size storage;

b) Measure the amount of inquiry traffic terminated at the FEP;

c) Run the other 494 model with only the amount of inquiry
    traffic measured from b) using an infinite size storage.
    Thus, all the inquiry traffic would be serviced by either
    U494.

2) Transactions wait for the availability of the half-duplex channel
    modeled as a single server in both directions.

3) After seizing the server the transaction joins the Host input
    queue. The time it joins the queue is recorded as t_i, to be
    used as the starting of the service time.

4) After reaching the head of the queue the transaction enters the
    detailed Host model. It is at this level where the various
    functions of the host are simulated in detail, such as scheduling,
    overlaying, peripheral access, CPU execution, etc. The trans-
    action is terminated, and an output message is generated as a
    function of the input transaction.

5) The output segment then joins an output queue, in preparation for
    transmission to the FEP. The time it joins the queue is recorded
    as t_o. The Host service time for that transaction is then defined
    to be $t_o - t_i$.

6) After seizing the channel server, the output segment enters the
    FEP storage buffers awaiting processing by the FEP modeled as a
    single server. If the storage is actually implemented in parti-
    tions dedicated to separate Hosts, the size of the storage should
    be equal to the size of the partition allocated to the Host being
    modeled.

7) The processing in the FEP is only that amount required to store
    the output segment on auxiliary storage, i.e., disk or tape.
    Once that is accomplished, the output is terminated.

The model is fairly simple and straightforward. The level of detail inside the detailed portion of the model is arbitrary and can be left up to the designer. It can start as a simple, gross model evolving into a more sophisticated program as simple functions are replaced with more elaborate subroutines. The only portion of the external environment required are those functions in the FEP that directly communicate with the Host. Those are the upstream output queues and the downstream storage buffers and single server processor.

4.3.3 The LDCN Front-End Processor Model

The FEP model in the context of the global network model is shown in Figure 4.8. It is by far the largest model of the three processor models. Using the methodology presented in Chapter 3, the FEP model is obtained by reducing and eliminating portions of the global network model. Concentrators are connected only to the FEP under consideration. Therefore, the number of high-speed lines has been cut in half. The FEP only communicates with three Hosts, consequently, the number of Hosts in the model has been again reduced by a factor of 2. In addition, the input queue at the Host has been removed. The service-time distributions for the Hosts in this model are obtained from the detailed Host model presented in the previous section. There, we measured the time before it entered the input queue. Therefore, in this model, the input queue is not needed. These Host service-time distributions should also be the ones that the FEP under consideration communicates with (i.e., when evaluating the traffic pattern for the ASO FEP, the ASO Host computer service-time distributions should be used.). In addition, the resources at the concentrator, buffers, queues, input lines, CPU processing power, etc., should be proportional to the number of terminals that are communicating with the particular FEP under consideration. The concentrator portion has also been modified by enbedding the traffic generator inside the concentrator. This eliminates the
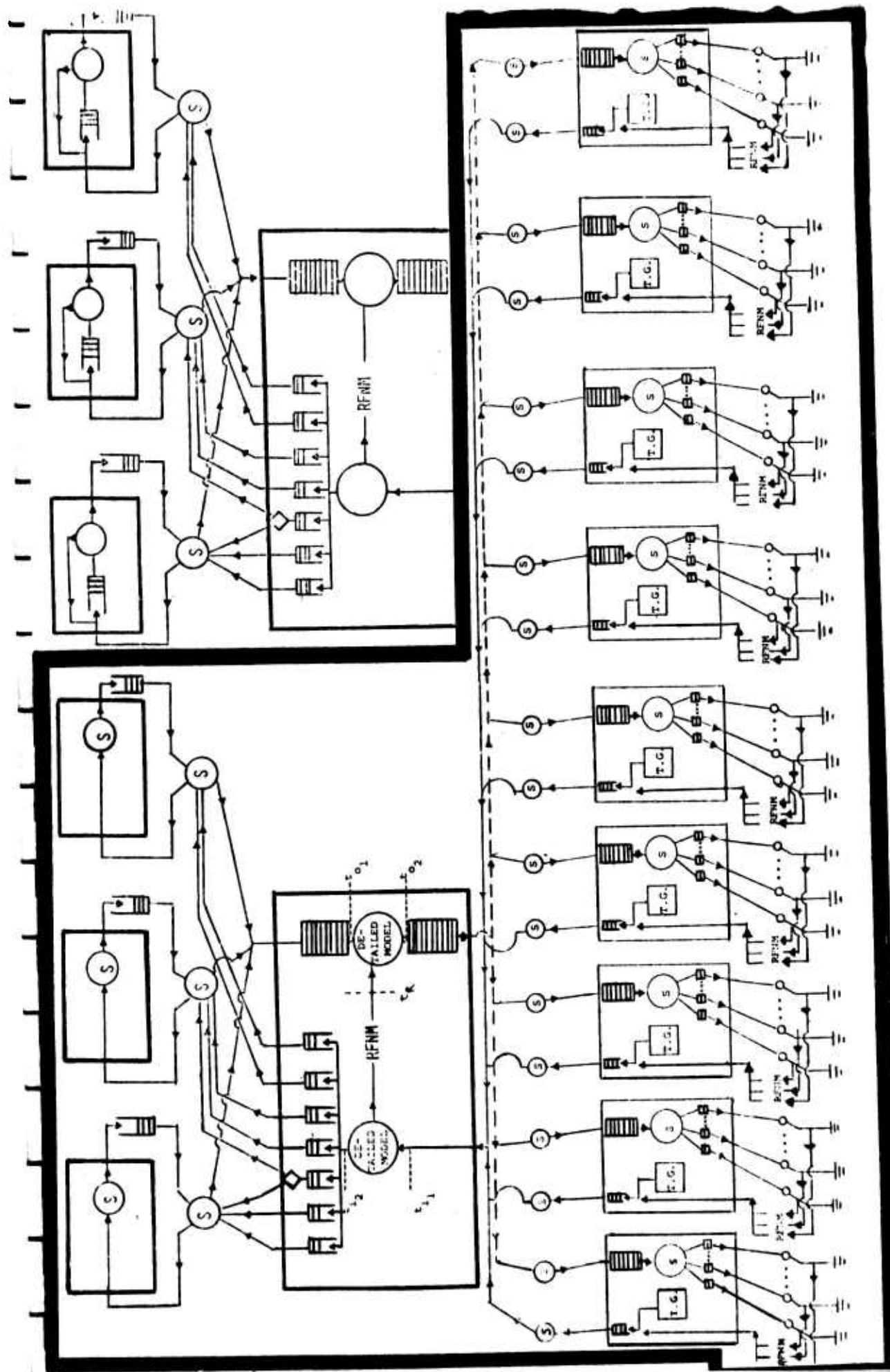
FIGURE 4.8: FRONT-END PROCESSOR MODEL

necessity of simulating the upstream concentrator server. All these changes are shown in more detail in Figure 4.9.

Again, using the modeling strategy described in Chapter 3, the simplified single server model of the FEP is replaced by the detailed version of the model. The dual-CPU configuration of the FEP is modeled in the detailed portion. Again we have represented the detailed portion as two servers; one in each direction. This was done only to point out that the service time distributions are typically different in each direction. The simulation program itself must take into account that there could possibly be only one CPU that is processing both the upstream and downstream traffic simultaneously.

If the simulation program for the FEP is too large, an alternative approach to modeling the concentrator (probably the largest component of the external environment) is possible. This is described below and depicted in Figure 4.10.

The life cycle of a typical transaction through this model is quite similar to that of the global network model. Therefore, we will only describe the differences.

1) As mentioned above, the traffic originates from within the concentrator. The traffic generator tags the transaction with the terminal identifier to be used on the downstream path later on. Again, the traffic pattern and load generated should correspond to that destined for the particular Hosts and, therefore, the particular FEP being evaluated.

2) As the transaction enters the detailed portion of the FEP model on the upstream path, the time is recorded as t_{i_1}.

3) The detailed upstream portion of the model allows simulating, as in the Host model, the various phases of processing involved. These include error-checking, CPU execution, the writing of batch
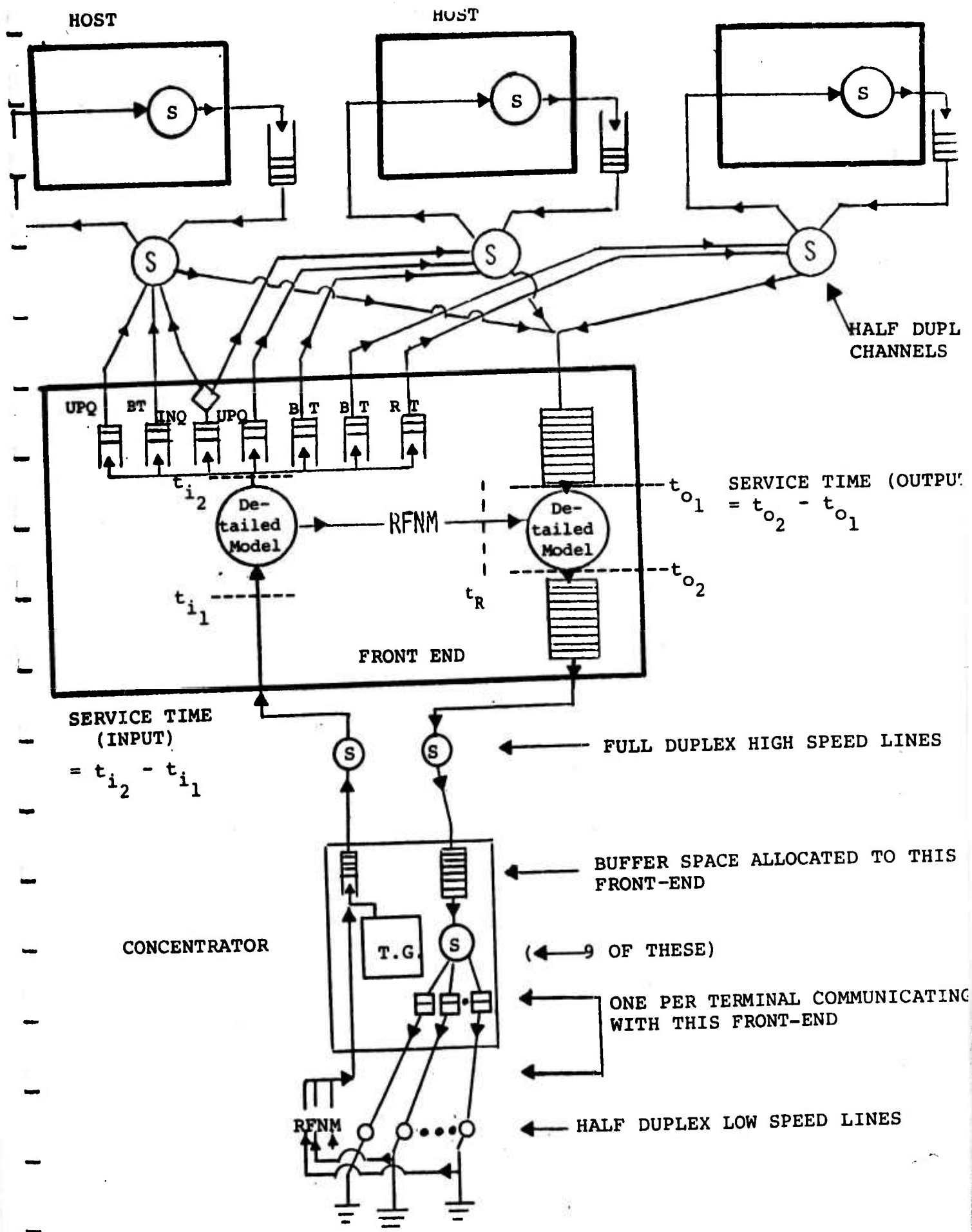
FIGURE 4.9:   FRONT-END PROCESSOR MODEL
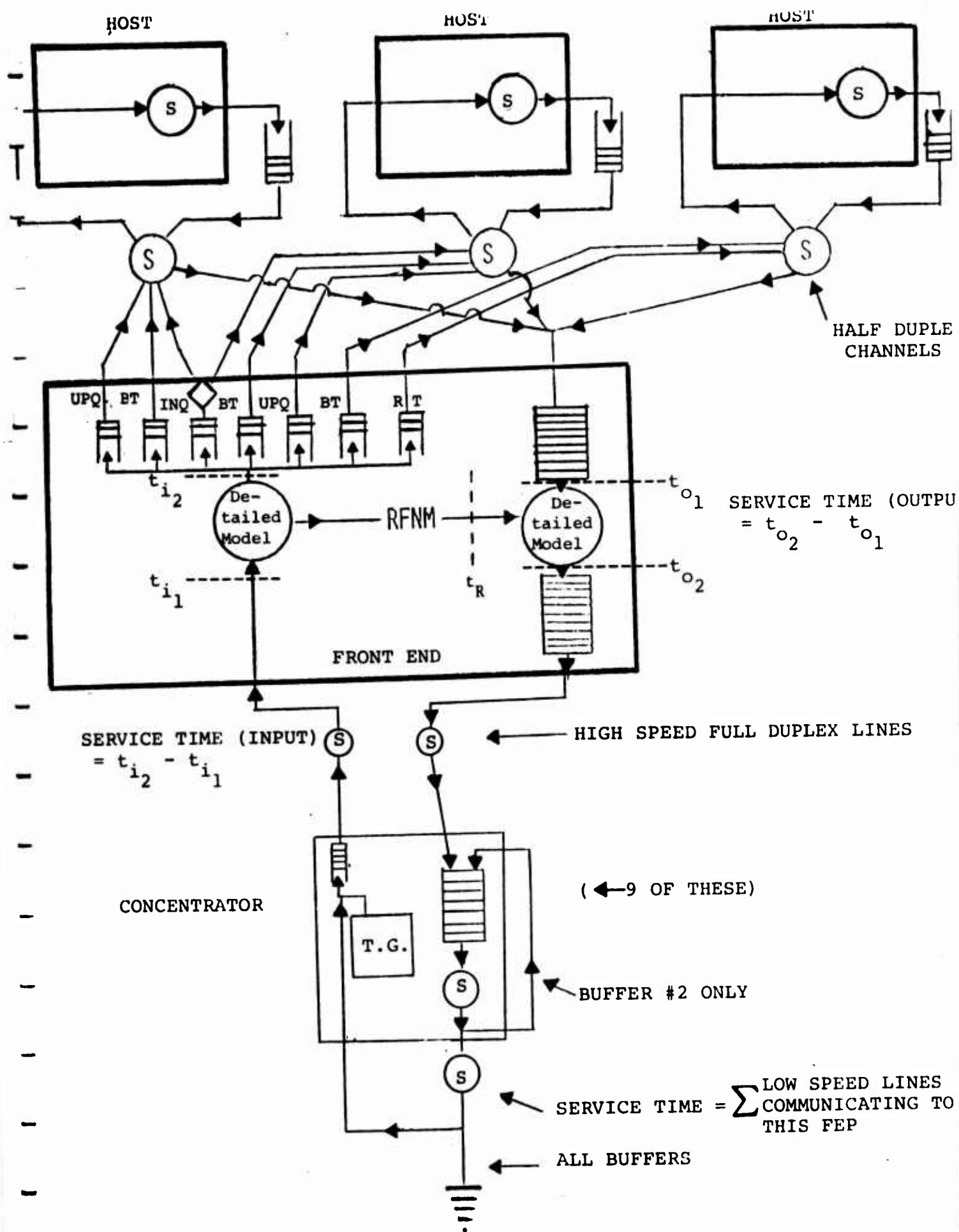
FIGURE 4.10: ALTERNATE FRONT-END PROCESSOR MODEL

transaction onto secondary storage, retrieving batch transaction from secondary storage, etc.

4) As the transaction leaves the detailed upstream portion, the time is recorded again, this time as t_{i_2}. The service time then for this transaction through the FEP in the upstream direction is $t_{i_2} - t_{i_1}$.

5) After the transaction is processed by the Host and an output is generated, it enters the downstream detailed portion of the FEP model via the half-duplex channel. Here again, the time is recorded as t_{o_1}. It is at this point that the functions and tasks performed in transmitting output to the concentrator by the FEP are simulated in detail. These include the storing and retrieving of output messages to and from secondary storage, processing involved in segmenting the output messages from the Hosts, processing the RFNM, interleaving and transmission of output segments to the concentrator, etc. Because the concentrators have no auxilliary storage, a scheme of double buffering is used to provide continuous transmission from the concentrator to the terminal. At the start of output, two output blocks are sent to the concentrator. As a buffer is emptied, a Request for Next Message (RFNM) is sent by the concentrator to the FEP. As the first two blocks leave the detailed portion of the downstream FEP model, the time is again recorded, this time as t_{o_2}. Therefore, the service time of the FEP for the storing of the entire output message and the retrieval of the first two blocks is $t_{o_2} - t_{o_1}$. The FEP service time for retrieving subsequent output blocks depends on when the RFNM is received by the FEP. Therefore, we define another time stamp t_R as the time the RFNM arrives and initiates retrieval of the next output block from storage. Therefore, the service time for output blocks other than the first two is equal to $t_{o_2} - t_R$.

6) Output blocks terminate after being served by the low-speed line server at which time the RFNM is generated. It is necessary to

4.20

include the low-speed terminal lines in this FEP model because the generation of the RFNM which will initiate FEP processing occurs after the transmission of an output block on the low-speed lines. This requirement could make the FEP program very large and costly to run. On the average only half of the total terminal population need be included in the program. (Recall that only the terminals communicating with the particular system and, therefore, the particular FEP are required.) This still (approximately 350) might limit the level of detail obtainable in the FEP program. In the next section we present an alternative scheme to resolve this problem.

An alternative approach to modeling the concentrator in the FEP program is shown in Figure 4.10. All the low-speed line servers are replaced by one single server with service time equal to the sum of the line speeds. This is not totally accurate in the low traffic case, but in the high to peak traffic load, this technique does return the average line speed service time. The goal is to remove the necessity of simulating every low-speed line in this model, therefore, reducing the size of the program. The above scheme does this for the lines themselves, but the double buffer mechanism still remains. That also can be replaced by the following sequence of changes:

1) Two priorities are associated with output blocks sent from the FEP to the concentrator. The higher priority is associated with the two buffer message that initiates output. The lower priority is associated with all single buffer messages that are initiated as a result of a RFNM.

2) After the two-buffer message is processed by the concentrator server, it is split into two messages. The first message seizes the low-speed line server and subsequently generates a RFNM and terminates. The second message has its priority lowered and is fed back in the queue for processing as a single buffer message. Only the first two output buffers are split in this manner.

4.21

Of course, this does not exactly model what is occurring in the concentrator. However, this approximation should be sufficient. The goal in this model is the detailed simulation of the FEP. Approximations in the external environment should be tolerable, since they lead to problem and program reductions, thereby allowing greater flexibility in the detailed model.

From the FEP model, we also define a system service time distribution which will be used by the detailed concentrator model. If we take the time interval $t_{o_2} - t_{i_1}$ for all messages in the FEP model, that defines the amount of time required to process an input message, create an output message and prepare it for transmission to the concentrator. This interval is the total turnaround time seen by the concentrator. This delay will, of course, depend on which system (ASO or SPCC) and which host at the system the transaction requires. Consequently, a substantial amount of statistics must be gathered from the FEP model that will be used in the concentrator model. These statistics are more fully described in Chapters 5, 6, and 7.

4.3.4 The LDCN Concentrator Model

The LDCN Concentrator Model is shown in Figures 4.11 and 4.12. The model is the same basic structure as in the global network model. However, the FEP and Hosts have been replaced by an Input/Output Function Box that given an input message or RFNM, generates an output according to the system service time distribution obtained from the FEP model described in the previous section. In addition, the simple single-server model of the concentrator is replaced by the more detailed version of the model. The life-cycle of a typical transaction is again similar to that in the global network model. The differences are described below:

1) The time an upstream message enters the detailed portion of the model is recorded as t_{i_1}. As it leaves the detailed portion, the time is recorded as t_{i_2}, thus attaining an input service time as $t_{i_2} - t_{i_1}$.
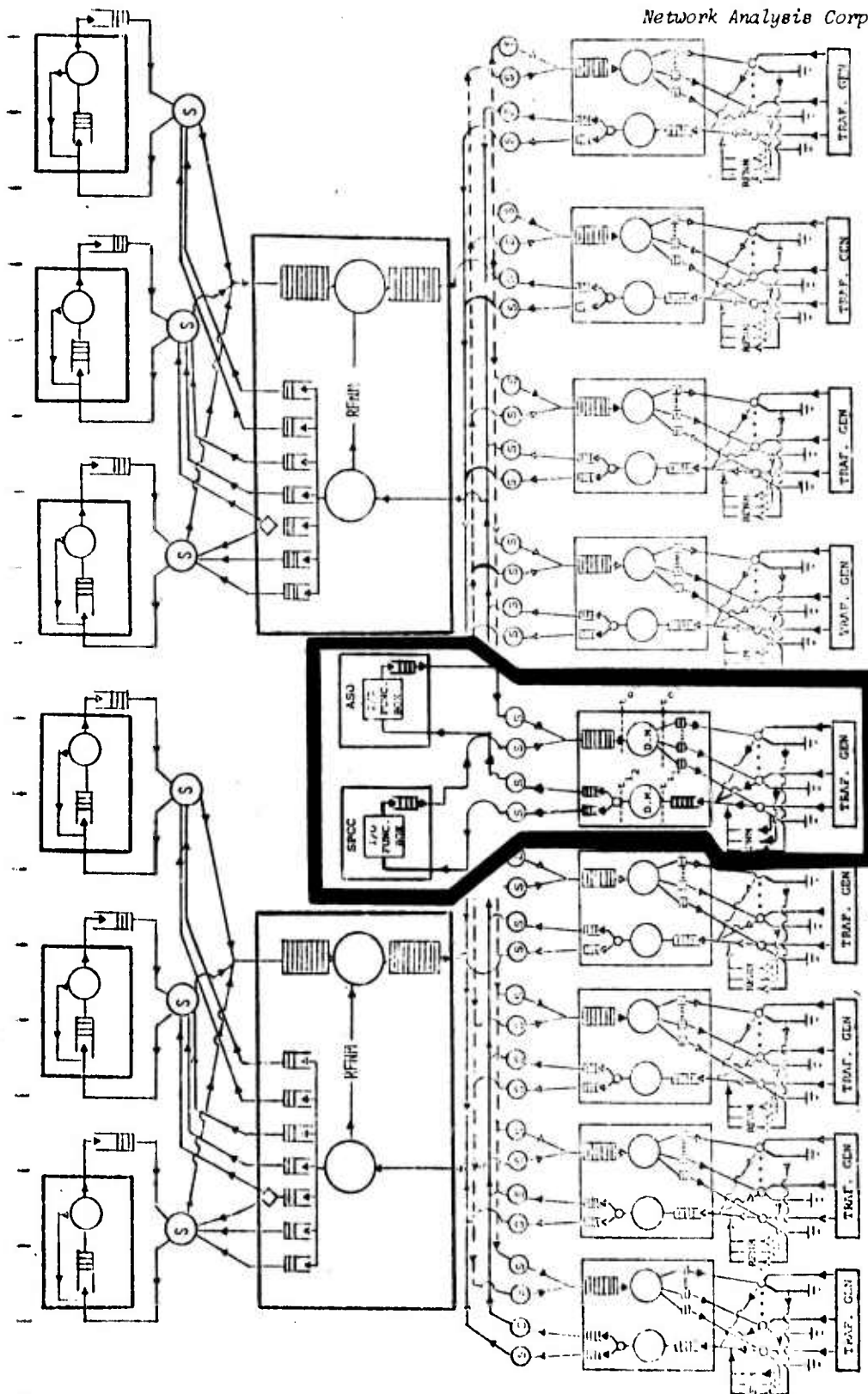
FIGURE 4.11: CONCENTRATOR MODEL

4.23
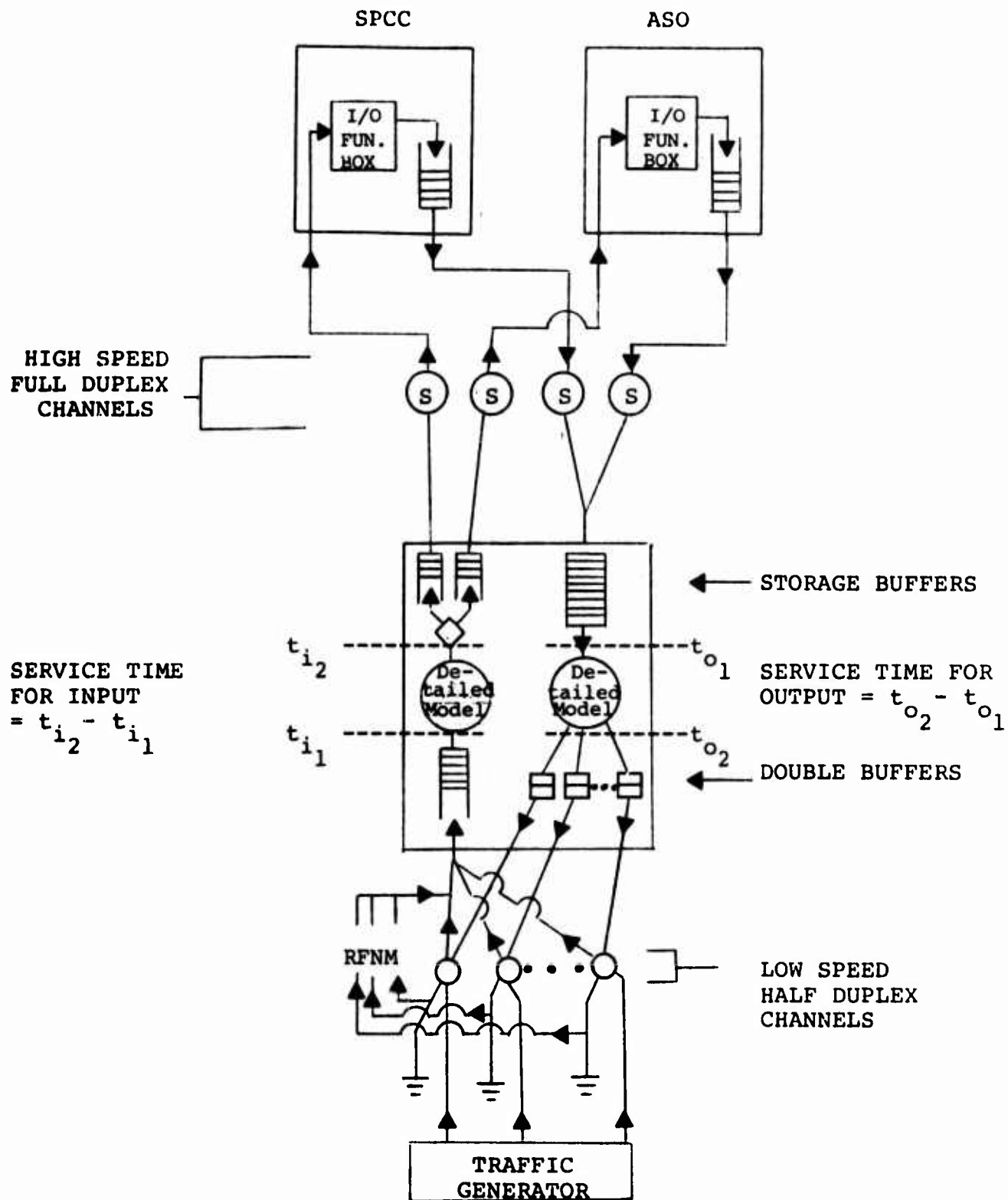
FIGURE 4.12:   CONCENTRATOR MODEL

2) After seizing the high-speed line server, the message enters a system input/output function box. This device models both the FEP and Host that the transaction would use. The service time is obtained from the distribution mentioned above.

3) As the output segments enter the detailed portion of the model, the time is recorded as t_{o1}. As it leaves, the time is recorded as t_{o2} giving a downstream service time of $t_{o2} - t_{o1}$.

Various experiments can be run using the basic concentrator model. The goal is to obtain characteristics of the concentrator delay for all nine concentrators. Typically the local access configuration (number and speed of the low-speed lines) at each concentrator will be different. In addition the traffic load and pattern will differ. The same program can be used for the experiments if the local access configuration is supplied as input. Service time distributions for the concentrator can be obtained for each configuration and used in the various other models. However, when a new feature is implemented in the concentrator, the corresponding simulation program must be run several, if not nine, separate times to verify that all concentrators can accommodate the new feature. However, when a change in the local access configuration of one concentrator occurs (a change that might happen frequently) if other concentrators have existing configuration similar to the new updated one, a new simulation run need not always be performed.

4.4 The Structure of the Detailed Portion of the LDCN Models

Up to now we have left open the structure of the detailed portion of each of the processor models. In this section we describe two alternative approaches that could be used and implemented in the simulation programs. Each technique has advantages and disadvantages which are also presented.

The first approach, and probably by far the most classical, is to code the specific functions and tasks directly into the program. The life-cycle or sequence of facilities inside the processor visited by a transaction are

enbedded inside the code. This method conveniently allows the functions
and tasks of the particular processor to be described and documented in
the simulation code. The program designer would be free to structure the
detailed portion as he wished, thereby allowing greater flexibility in
choosing the areas of most concern and desirable levels of detail. How-
ever, three separate program modules must exist and be maintained. In
addition, every change in the life-cycle of a transaction implies a change
in the code, a re-compilation of the module or subroutine, and a reloading
of the entire program.

The other approach is what can be referred to as the "unified" method
of processor modeling. It is based on the principle that every transaction
in a computer system goes from one "server" to another in an orderly and
predetermined fashion until its processing is completed and/or exits from
the system. This sequence of events or "cycle-vector" would be different
for each processor. However, if the detailed portion of each of the pro-
cessor models were structured as the same general flexible skeleton, the
cycle-vector for each processor could be read in as input and thereby
create different "programs." This concept avoids recompiliation and re-
loading of the simulation program. Only one version of the program need
be designed and coded. However, there are some disadvantages to this ap-
proach. There is an overhead involved in reading in the cycle-vector and
initializing data structures in such a program. Design runs on a fixed
model would run longer and cost more with this approach than the first.
The input is more complicated, possibly requiring a pre-processor for
verification. Also, a general program, to accommodate all possible fea-
tures, might be so large that other features specific to each particular
processor might have to be reduced or removed. However, a well-designed
program structured in this manner could be both general and flexible in
addition to being cost effective.

The tradeoffs of each approach must be made at implementation time
and in conjunction with the various simulation languages. Certain high
level languages that provide desirable features might be restrictive in
terms of the second method. Other more primitive languages that allow

extensive I/O might not provide the capabilities of others. A more detailed description of simulation languages and program implementation is presented in Chapter 8.

5. INPUT TO SIMULATION PROGRAMS

A general modeling program for a communications system should reflect three fundamental dimensions in which variations may be examined:

- System configuration,

- Processing cycles of messages, i.e., software design,

- Demand and service characteristics.

These dimensions may be viewed as a hierarchical structure. The hardware configuration of a communications system is usually viewed as the most fundamental aspect of the system. For a given configuration, several software designs may be examined. Each such design is characterized by the processing cycles of the messages. Finally, for each hardware and software design, performance is appraised for a range of traffic characteristics and service times of the hardware and software elements.

5.1 System Configuration

The system configuration inputs are divided into two main levels: network topology defining the overall system structure, and link and node model attributes providing details for each structural component. The basic structure of the configuration data base is shown in Figure 5.1. Note that the data structure has provision for information for network analysis beyond that required for network modeling. This permits a common input front end to serve network performance modeling and topological design data bases. Each simulation program extracts only the required information needed as input. As output is generated, such as the service time distributions for the aggregate model attributes, it is either substituted for the old, outdated information or is added as another statistic generated for a different set of conditions (e.g., a different traffic pattern or load).

1. **Network Topology**

    1.1 **Nodes (N)**

        1.1.1 Node i.d.

        1.1.2 Node Descriptors

        1.1.3 Position Data (only if other than operational modeling desired)

    1.2 **Links (L)**

        1.2.1 Link i.d.

        1.2.2 Link Descriptors

            1.2.2.1 Node at First End

            1.2.2.2 Node at Second End

        1.2.3 Tariff Data (only if other than operational modeling desired)

2. **Link Attributes (L#)**

    2.1 Nominal Line Speed

    2.2 Error Rate

    2.3 Half or Full Duplex

3. **Node Models (N#)**

    3.1 Aggregate Model Attributes (A)

        3.1.1 Priority

        3.1.2 Service Time Distribution

        3.1.3 Average Queueing Delay

    3.2 Detailed Model Attributes (D)

        3.2.1 CPU

            3.2.1.X CPU attributes

        3.2.2 Peripherals

            3.2.2.X Peripheral Attributes

        3.2.3 Memory

            3.2.3.X Memory Attributes

        3.2.4 Buffers

            3.2.4.X Buffer Attributes

        3.2.5 Communications Interfaces

            3.2.5.X Communications Interface Attributes

Figure 5.1 - SYSTEM CONFIGURATION INPUT

5.2 Software Design

The software design of the system is defined by the processing cycles of the messages. The processing cycle of a message is specified by the sequence of servers visited by the message, and the state the transaction induces in the server at each stage. As mentioned in Section 4.4, two alternatives exist in modeling the software design of a communications processor. The first is to imbed the software functions and tasks directly into the simulation program code. This method requires no input but necessitates recompilation and reloading of the program if a different software design is investigated.

The other alternative is to have the software design, i.e., the processing cycle of messages, read in as input. Thus, a state vector is defined for each server (or facility), where each distinct state is characterized by interrupt time distribution, service time distribution, and priority. For each message type, the sequence of servers and their appropriate states are then defined in a cycle vector. The state and cycle vectors provide the linkage with the facilities to interconnect the queueing structure in the detailed processor model. The basic input structure is shown in Figure 5.2. For the global network model software considerations are not too important. As a consequence, the state and cycle vectors would not be used there.

The use of message processing cycles is a particularly convenient means of specifying the system operation and structure. Most designers frequently think in terms of the "life cycle" of a message, that is, the sequence of events followed by a message as it moves through the system. Such a life cycle is illustrated by the extremely simplified sequence of events listed below:

1) Transmission of a message to an input buffer,

2) (After being queued) processing by the CPU,

3) (After being queued) disc access,

5.3

```
┌──────────────────────────────────────────────────────────────────────┐
│  1.  Message Type                                                      │
├──────────────────────────────────────────────────────────────────────┤
│  2.  Cycle Vector                                                      │
│                                                                        │
│      2.1  Facility i.d.                                                │
│                                                                        │
│           2.1.1  State Type                                            │
│                                                                        │
│           2.1.2  State Attributes                                      │
│                     :                                                  │
│                     :                                                  │
│                                                                        │
│      2.2  Facility i.d.                                                │
│                     :                                                  │
│                     :                                                  │
├──────────────────────────────────────────────────────────────────────┤
│  3.  State Vectors                                                     │
│                                                                        │
│      3.1  Facility i.d.                                                │
│                                                                        │
│           3.1.1  State i.d.                                            │
│                                                                        │
│           3.1.2  Priority                                              │
│                                                                        │
│           3.1.3  Pre-Emptive/Non-Preemptive                           │
│                                                                        │
│           3.1.4  Cycle Stealing Rate                                   │
│                                                                        │
│           3.1.5  Interrupt Processing                                  │
│                                                                        │
│           3.1.6  Processing Time Distribution                         │
│                                                                        │
│           3.1.7  Operational Functions Subroutines (e.g., Protocol)    │
│                     :                                                  │
│                     :                                                  │
│                                                                        │
│           3.1.X  State i.d.                                            │
│                                                                        │
│      3.2  Facility i.d.                                                │
│       :                                                                │
│       :                                                                │
└──────────────────────────────────────────────────────────────────────┘
```

Figure 5.2 - SOFTWARE DESIGN INPUT

4) Processing by CPU,

5) (After being queued at an output buffer) transmission of the reply on an output transmission line.

There is almost a direct translation from the "life cycle," illustrated above, and the processing cycle needed for specification in the model. Only refinements in modeling are needed, and are easily developed. For example, from the above illustration, at the beginning of the event "1," an "interrupt" would advance the simulated clock while delaying the completion of all the messages already in the CPU. As the input message is being transmitted, the impact of character interrupts can be approximated by reducing the CPU processing power by a certain "percentage." Thus, the impact of event "1" on other messages can be given in terms of "interrupt" time and "percentage" CPU processing power degradation. The impact of the other four events can be described in a similar fashion.

The incorporation of system control logic is handled by subroutines triggered from the state vector processing. Thus, such logic can easily be excercised at the discretion of the designer.

5.3 Demand and Service Characteristics

The third level of the hierarchy is the demand and service characteristics. Most modeling exercises are not to determine system behavior for only a single set of conditions, but rather for a range of possible conditions. A general program must be easily changeable to reflect different conditions. This is accomplished by specifying the conditions through a set of parameters.

The basic parametric structure is shown in Figure 5.3.

1. **Message Type Attributes**
1.1 Type
1.2 Priority
1.3 Message Length Distribution
2. **Arrival Patterns**
2.1 Scheduled
2.2 Random
2.2.1 Interrival Distribution
3. **Traffic Mix**
3.1 Message Type
3.2 Percent of Total Traffic
4. **Traffic Matrix**
4.1 Message Type
4.2 Percent of Type
4.3 Origin
4.4 Destination
5. **Traffic Levels**
5.1 Lowest Level
5.2 Highest Level

Figure 5.3

SERVICE AND DEMAND PARAMETER INPUT

5.4 Input to the Specific LDCN Models

5.4.1 Input to the Global Network Model

The basic input to the global network model is the system config-
uration, consisting of network topology, and aggregate nodal attributes.
In particular, the following inputs are identified:

1) For each Host –

   a) Service time distributions

2) For each FEP –

   a) Service time distributions for both upstream and downstream
       directions

   b) Interconnections to Hosts

   c) Output buffer storage size

3) For each Concentrator –

   a) Service time distributions for both upstream and downstream
       directions

   b) Interconnections to FEPs

   c) Output buffer storage size

   d) Local access configurations of terminal lines

It is at the concentrator level where the traffic load and pattern
characteristics are required for input.

5.7

5.4.2 Input to the Host Model

The input to the Host program can be divided into 2 parts: those that deal with the Host itself, and the others that deal with the external environment, namely the FEP.

The system configuration input, relative to the Host, consists of the hardware configuration at the Host. These include memory, peripherals, etc.

The software design input for the Host consists of the state and cycle vectors describing the various operating systems implemented in or under consideration for the particular Host.

The input to the Host model that pertains to the FEP consists of configuration data, software considerations, and service and demand characteristics. Of course, the speed of the channel connecting the Host and FEP must be specified. In addition, the output buffer storage size, which is used as a throttling mechanism, should be input. Obviously this should not be the total amount of storage available at the FEP. It should reflect the amount available to the particular Host. It is either the amount dedicated to that Host (if partitions are implemented) or proportional to the amount of traffic processed by the FEP for this particular Host (dynamic implementation). For U494 Host models, the size of the inquiry storage mechanism (used for the shared transactions) must also be specified.

The nominal service time distribution to process the output message (remove from primary storage and store on secondary storage) is required as input.

The traffic that arrives at the Host originates at the FEP. The traffic generator in the FEP will generate traffic according to input parameters. These include distributions by transaction type (inquiry, upquiry, batch) and size. For batch transactions, a utilization criterion for acceptance by the Host must be specified.

If aggregate traffic statistics for a particular Host are not available from measurements, a traffic preprocessor can be applied to the traffic loads and patterns used at each concentrator to extract only the traffic destined for that particular Host.

5.4.3 Input to the FEP Model

As in the Host model, the input to the FEP model can be divided into 2 areas: The FEP itself and the external environment. This time the external environment is composed of two classes of devices: Hosts and concentrators.

For the FEP, the input consists of the hardware configuration and the software design. The hardware input includes amount of main memory, number and speed of peripherals, and amount of secondary storage. The state and cycle vectors describe the software implemented in or under consideration for the FEP. Some of the software-related functions and tasks that can be evaluated are as follows:

a) Various queueing disciplines,

b) Dual - CPU servicing disciplines, cutover thresholds,

c) Overhead involved in gathering network statistics,

d) Whether statistics gathering is in effect or not.

There are various inputs required for the external environment of the FEP. The number and identity of the Hosts and how they are connected to the FEP must be input. For each Host, the service time distribution is required. The number and identity of the concentrators and the line speeds of the connections are required input. A full description of the configuration of the concentrator is necessary including the number and speed of the local access lines and the percentage of resources available at the concentrator for the particular FEP under evaluation. The down-stream service time distribution of the concentrator is needed. The concentrators are the source of the traffic for the FEP. Therefore, it

is here that the traffic load and pattern distributions are specified. Recall that these traffic statistics should only be for the FEP under evaluation (ASO or SPCC). Again a preprocessor could be used to extract this information from the traffic data base at each concentrator.

5.4.4 Input to the Concentrator Model

The input to the concentrator model consists mainly of input describing the concentrator. The only input for the external environment is the _system_ service time distributions obtained from the FEP model.

At the concentrator, the hardware configuration which includes the communication interfaces and the number and speed of the lines, both to the terminals and the FEPs are required for input. The state and cycle vectors defining the software design in the concentrator are input.

In addition, the traffic load and pattern for the local configuration, which drives the concentrator model is required input. These, however, are typically different for each concentrator. However, the same program is used; only the input is different.

6. OUTPUT

The principal output of the simulation programs is the statistical information describing system or processor behavior. In addition to the statistics obtained from the simulation, the report generator should provide a description of the network or processor configuration being evaluated plus a listing of all input parameters for easy cross-referencing and verification. Also, when the tracing of a message option is in effect, that output will aid both designers in attaining insights to the system as well as providing debugging facilities for the programmer at implementation time and for future updates.

The statistics output by the report generator that are common to all the LDCN models include:

1) For each server or facility (e.g., communications line, processor, or facility within a processor (CPU, memory, peripheral etc.))

   a) utilization

   b) number of times occupied

   c) average time per occupation or service time

   d) availability

2) For each queue

   a) maximum and average contents

   b) total and percent of entries that did not have to wait (zero entries)

   c) average queueing time for all and only-delayed transactions

d) buffer space used by queues

3) For each message type

a) the average lifetime in the system (i.e. round-
trip delay)

b) the maximum and minimum observed lifetimes

c) the number of messages generated and serviced
by the processors.

In the detailed processor models, the software design is evaluated by
examining the state and cycle vectors. Therefore, additional output
includes:

4) For each state in each cycle vector

a) the average time spent in each state

b) the number of times entering a particular
state

c) maximum and minimum time spent in each state

Flexibility exists in the form and style of the simulation output.
Some high level simulation languages automatically provide some form of re-
port generators and plotting capabilities. If these are not sufficient,
programs can be written that can present the output in any manner desired.

The report generator provides several options to the user, ranging from
simple summary reports to extensive detailed reports. The model user has
the option of specifying retention of only needed information or all infor-
mation resulting from exercising the model. This allows efficient model
execution when only particular information is desired. Exercises can be

6.2

tracked by summary reports with the total information retained for presentation in detailed reports at a later time.

The above statistics provide the systems designer with the information necessary to evaluate the performance of the processors and the entire network. Because the modeling strategy presented relies on the interaction between the various simulation programs, additional output is required to serve as input for other programs. We have identified the linkage information to be the service time distributions for the various processors. These distributions would be for a fixed set of conditions; traffic load, mix and pattern but would be a function of the message size (both input and output) and message type. This information is available from the set of statistics presented above, and in fact, would provide the designer with further insights as well as serving as the linkage between the simulation programs. The form and methodology of this linkage information is described in the next chapter.

7. LINKAGE METHODOLOGY

In Chapter 3 we presented a modeling strategy that allows global net-
work modeling as well as detailed processor modeling. Because of the bi-
directional nature of communications processors, it was identified that
for processor modeling some form of the external environment need also
be simulated at the same time. The external processors were to be modeled
as simple single servers with service times according to service-time
distributions. These distributions were to be obtained from the output of
other detailed processor models and also be used in the global network
model. In this chapter we describe a methodology for obtaining this link-
age information and its use in the detailed models.

7.1 Linkage Form

The information that links the LDCN models together are the service-
time distributions of each of the processors. As stated in the previous
chapter, these distributions are a function of message type and size.
This output is obtained from the report generators for the user and is
usually in the form of a histogram or an approximated continuous curve.
It is now necessary to transform this information into machine-readable
form to serve as input for the simulation programs that require it. This
can either be in punched-card format, kept on-line in a data set or file,
or presented in a manner that can be transcribed by hand and then input
manually. Post-processing routines can perform this task at the end of a
program run and create the appropriate form of input. Alternatively, pre-
processing modules can take the raw output data from one program and ex-
tract the required information. A third option is to create a stand-alone
program that performs this task. These options are shown in Figure 7.1.
The decision as to which scheme is the best depends on how the report
generators are implemented. One can take advantage of how the other
statistical output is generated by obtaining this linkage information in
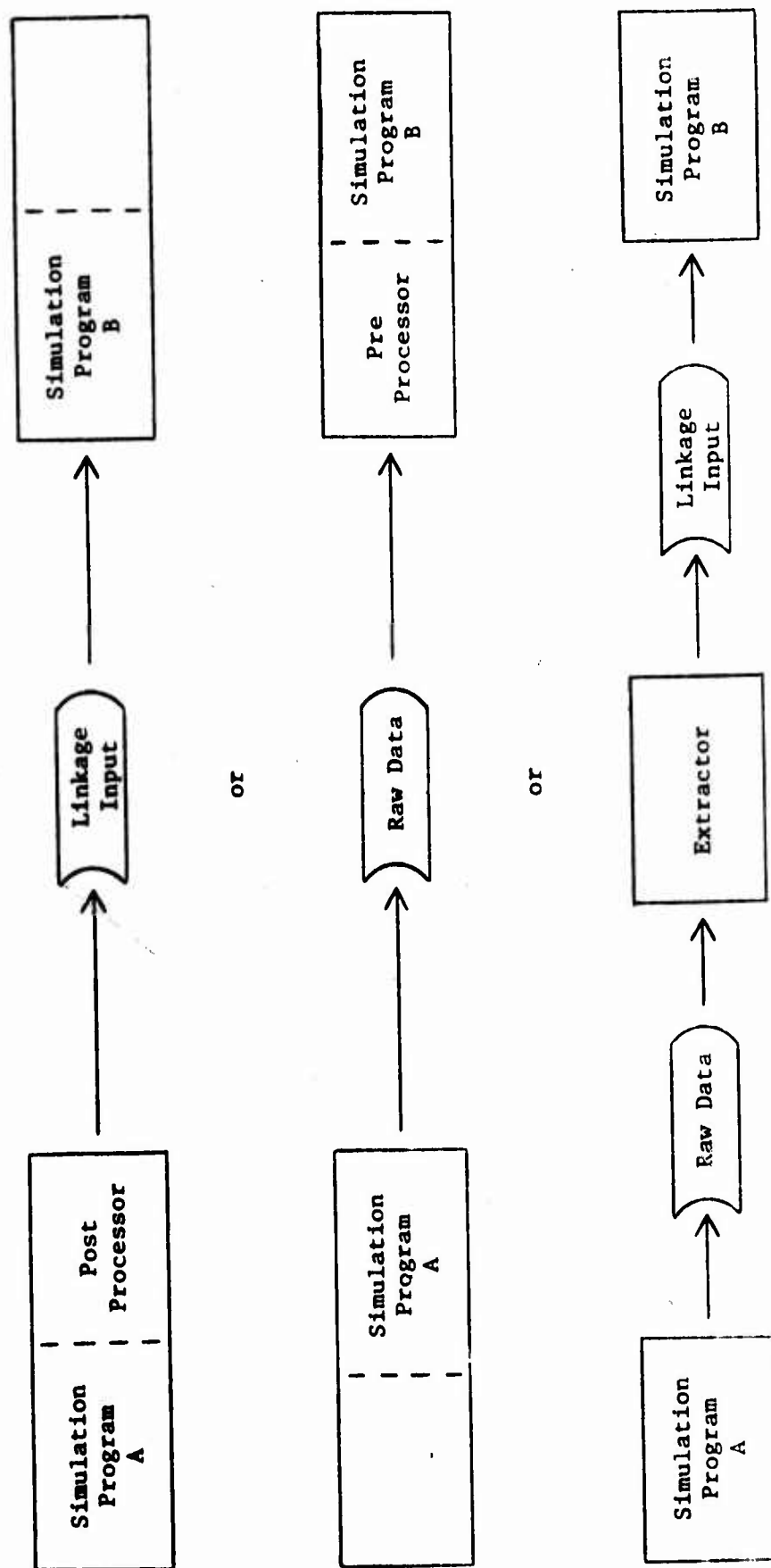the same manner.

Figure 7.1

PROGRAM LINKAGE OPTIONS

7.2 Methodology for Using the Linkage Information in the LDCN Models

The global LDCN network model provides no linkage information but uses those of the other detailed models. At first, since the detailed models are not built, simple routines can be written to read in the gross service-time distributions of the processors. As more accurate distributions become available from the detailed models, this linkage information must be available to the global network model.

Each detailed model also requires the service-time distributions of other processors for input as well as providing it on output. The order in which these distributions are obtained is important in attaining meaningful, accurate results and avoiding repeated simulation runs for a fixed set of conditions.

The recommended sequence of simulation runs for the LDCN detailed models under a fixed set of conditions (traffic load, pattern, and mix) is as follows:

1) Run the Host model using the nominal FEP output service time;

2) Using the accurate Host service-time distribution from 1) and the nominal concentrator output service time, run the detailed FEP model;

3) Using the system service time distribution obtained from the FEP model, run the detailed concentrator model;

4) Compare the new service time distribution obtained for the concentrator with that distribution used in the FEP model. If they are equivalent, then stop;

5) Otherwise, rerun the FEP model using the new concentrator service time distribution and go to 3).

It is only necessary to simulate the Host only once for a given set of conditions. The service-time distribution of the Host will not depend on how fast or slow the FEP serves the output messages. The output queue at the Host might grow very large, but the service time, which we have defined as the time in the Host until being placed on the queue, will not change.

The FEP and concentrator models, however, rely heavily on each other. Of interest is the conjecture - is there any direct relationship between the service times of the FEP and concentrator? If one changes, does the other change and thereby cause a change in the first?

The sequence of steps shown above will detect this as well as validate the linkage information between the models. Oscillation between the concentrator service-time distributions used as linkage input to the FEP model and that actually obtained from the concentrator model can be expected. However, continual oscillation indicates an unstable relationship between the two processors. The results from the simulation runs should provide insight into this problem. Oscillations that converge rapidly indicate that only the preliminary assumptions were incorrect.

8. IMPLEMENTATION CONSIDERATIONS

Choosing the appropriate simulation language for the LDCN models de-
pends on many elements. The general considerations include availability,
procurement cost, operating cost, training cost, portability, support, etc.
Other considerations pertain to the effectiveness of each language to solve
the various coding problems posed by the models proposed in the previous
chapters. In this chapter we present a comparison of the three primary
languages used for simulation programs: FORTRAN, General Purpose
Simulation System (GPSS) and SIMSCRIPT II. We also describe a relatively
new language, ECSS II, that was developed particularly for modeling computer
systems. We then describe the advantages and disadvantages each language
exhibits when actually applied to the LDCN models and modeling strategy.

Reitman[1] provides an excellent review and comparison of the three
most widely used simulation languages: FORTRAN, GPSS, and SIMSCRIPT II.
The comments are grouped into four basic categories: short term results,
ability of the simulation to represent the real world, long term results,
and effort required to use the language. A synthesis of his review is
presented below.

A. Procedure Oriented Languages - (FORTRAN, ALGOL, PL/1) They are
    not really simulation languages, but more like mathematical pro-
    gramming languages. However, they have been used for simulation.

    1) Short term results

        ● programmer must have good background in the language be-
          fore it could be used for simulation

        ● have to provide the simulation structure; none exists

        ● statistics gathering functions internally have to be
          structured

        ● not very flexible - revisions in complex systems require
          major modifications

- no graphic capabilities built in, additional programming
  involved

2) Ability of simulation to represent the real world

- could model almost any real-world situation

- the more complex systems require much greater effort

- mathematical capabilities are excellent, many special-
  purpose techniques for data smoothing and linear
  programming

- list processing is weak. Any simulation program requires
  some form of list processing to structure the model,
  some form must be provided for these languages

- maximum program size is flexible; overlays are possible

3) Long term results

- language generality, supported universally

- documentation left up to programmer, cross references
  available

- system designers other than original program should be able
  to follow the logic and detail of the simulation

4) Effort required

- considerable, but several programmers can work on the simu-
  lation in parallel if conventions governing the exchange
  between subroutines is specified in advance

B. GPSS - the first real programming language geared specifically for simulation

1) Short term results

- geared to get results quickly

- simulation structure is strongly evident

- many built-in features, format, organization and diagnostics

- statistics are maintained automatically both during and at conclusion of the simulation

- flexible; easy to change logic, data and results selected

- debugging aids are many

- graphic presentation of output is available

2) Ability to represent the real world

- desired level can be obtained

- can use byte, half-word or full-word arrays

- logical situations are well represented by Boolean equations

- mathematical capability is adequate for problems that do not require complex equations. However, the fact that values are stored as integers causes scaling problems and loss of precision in arithmetic computations such as division

- list processing is available, allowing FIFO, LIFO or any priority structures

- Maximum size of program is the tradeoff between available core storage and execution time. Overlays are possible

3) Long term results

- benefits from the highly structured language

- documentation capability is very good, comments can be part of every statement

- transferability of GPSS programs is excellent

4) Effort required

- GPSS provides the most working model for the effort expended

C. SIMSCRIPT II - programming system developed by RAND designed particularly for simulation

1) Short term results

- programmer should be compentent in SIMSCRIPT II

- no inherent structure for the simulation. Consequently, an extensive problem definition and structure should be developed before coding the model.

- relationships are through the entity - attribute - set relationship

- statistics obtained either during or after model execution are programmed by the system designer

- the language allows access to anything at any time, but the structure and format of the statistics has to be specified by the user

- flexibility is tied in with the basic subroutine structure of the programming approach. Individual subroutines can be compiled and added or substituted into the model. The same model may be run with different input data.

- programs to display output in graphic form are not built in, they have to be coded

2) Ability to represent the real world

- logical situations can be well represented with the Boolean capability. Very complex situations can be structured.

- mathematical capabilities will depend on the particular installation since SIMSCRIPT II is a separate programming system. Libraries of utilities will have to be developed.

- list processing capabilities are strong, owing to the structured data storage system. FIFO, LIFO and any priority structures are easily developed

- maximum program size again, depends on available computer storage. SIMSCRIPT II is a compiler language. As such, the compiler might use up an excess amount of storage. Overlays are possible.

3) Long-term results

- syntax of language is almost readable English

- as a consequence, the documentation throughout the program is excellent. Additional comments may be placed anywhere. Users other than the original programmer should be able to follow the model in detail easily.

- however, the structure of the program is still up to the programmer. For complex problems the readability does not describe the relationships between different factors. Simultaneous events are difficult to document. Changing the logic of an existing model is difficult for those changes which require restructuring of data or system attributes.

4) Effort required

- comparable to GPSS, yet the more complex models require con siderable more effort

More recently, work has centered on creating simulation languages designed specifically for modeling computer systems. There are now several languages available that allow a programmer the freedom to write his program referencing common computer hardware and software terms as part of the code. Typically the language is implemented as a preprocessor to a general-purpose simulation language. One such example is the Extendable Computer System Simulator II (ECSS-II) developed at RAND. ECSS-II is based on the SIMSCRIPT II language, with that language included as a subset. Consequently, all the advantages of SIMSCRIPT II are present in ECSS-II, plus some of the inadequicies are removed. In [2], Kosy states that:

"[ECSS-II] provides a rich variety of statements and data-structures for describing common computer hardware configurations, software operations, and workload characteristics in a natural and straightforward notation. Using these statements, one can compactly express, for example, the name, quantity, and per-formance of each kind of simulated hardware device, the behavior and resource requirements of each kind of job to be processed, the policies by which resources are assigned to jobs, the characteristics of messages sent through I/O devices within the model, and how the simulated system is to be loaded by jobs and messages from its environment."

The authors of ECSS-II have taken advantage of SIMSCRIPT II's inherent readability to create a language that utilizes computer hardware and soft ware jargon.

In addition, the added simulation structure provided by ECSS-II is directly applicable to modeling computer systems. In an early interim report comparing ECSS-II (then, just ECSS) with FORTRAN and PL/1 [3], Kosy highlights these added structural features:

> "To this [SIMSCRIPT II] base ECSS adds four new elements to describe the statics and dynamics of computer systems for simulation: the System Description, Load Description, the Service Routines, and an extension to the preamble called the Definition Description.
>
> The System Description consists of a group of declarative statements that specify the number of each type of device in the system, the names of the devices, the characteristics of each device, and how this hardware is interconnected. One can define CUPs, core storages, terminals, disks, or any other kind of device in terms of its data transmission capabilities, instruction execution rates, and storage capacity. In general, these hardware elements are the resources to be allocated and utilized during the simulation.
>
> In the Load Description section, special routines called "jobs" are defined to describe a system's dynamic behavior. Jobs simulate the work of real application and control programs by indicating sequences of hardware utilization commands. These commands are used, for example, to indicate amounts of data transmission and instruction execution, to get and free simulated storage space, to define conditional delays, to start and terminate jobs, and so on. Jobs are processes—simulated time advances as a job is executed and many different changes to the state of the system are usually included in one job. Quite detailed representations of computer program behavior can be described by intermixing ECSS and SIMSCRIPT commands within the jobs. Jobs and events can be used together in the same model to provide an extremely powerful composite world-view of system dynamics.
>
> The Service Routines are a collection of SIMSCRIPT II routines which implement the Load Description commands. They assume the details of job processing and time advance, as well as updating the variables that define the state of the system as jobs interact with devices. Also incorporated into the Service Routines are a number of resource-management algorithms that provide a kind of built-in operating system. This capability gives the user the power to specify multiprogramming, contiguous-storage management, conversational messages and other high-level activities with only a few statements.
>
> The Definition Description is of lesser importance than the previous three elements, but it does supply the user the ability to define his own commands, to use his own terminology for certain computer-related dimensions (bytes for

transmission, say, or pages for core space), and a statement
for compact definition of table functions, all of which are
lacking in SIMSCRIPT II.

We now describe some of the advantages and disadvantages of each of
the above languages when applied to the specific application at hand;
i.e., the LDCN models.

FORTRAN, because it is not a simulation language, is probable the
least attractive for such a large simulation project. Considerable effort
would have to be expended in providing even the basic simulation structure.
List processing capabilities, essential to any complex simulation program,
are generally weak; however, we are aware of some available list pro-
cessing packages for FORTRAN programs. The I/O capabilities of FORTRAN
are probably the best. The software design of the processors could be
implemented using the cycle and state vector approach and input to a
skeleton program that could be used for each of the detailed processor
programs. Also the I/O of the linkage data would be easier in FORTRAN.
The previously delivered stand-alone concentrator simulator was designed
with the vector approach and written in FORTRAN. The language is a
viable one but there are better ones.

GPSS, because it is a simulation language, would be better than
FORTRAN. However, because of the general purpose nature of the language,
it may still not be the best language to use. Limited I/O facilities
prohibit using the vector approach for evaluation of various software
designs in the processors. There is no inherent way to selectively
change only portions of the program without "recompiling" the entire
program. However, the diagnostics and queuing structures are strong.
Since the LDCN models rely heavily on queues, the programming effort
might be reduced. In addition, the U494 stand-alone Host simulator
was written in GPSS and existing FMSO personnel are proficient in GPSS.

SIMSCRIPT II is probably the better of the two simulation languages.
Complex models, of which the LDCN models are ones, are more flexable
when written in SIMSCRIPT II than in GPSS. Limited I/O capabilities,
again, limit the use of cycle and state vectors for defining software

design in the processors. However, separate modules or subroutines that handle the software implementation could be changed and recompiled separately. But it might be difficult to structure the programs in this manner (all the software functions in one or two routines). In addition, personnel familiar with SIMSCRIPT II might be limited.

ECSS-II is a relatively new and untested language. However, the syntax is geared specifically towards computer systems modeling, which is the problem faced here. It conveniently allows both hardware and software evaluation. It has all the capabilities of SIMSCRIPT II plus developments have been made to improve the deficiencies (e.g., statistical instrumentation)[4]. It might not provide the level of detail desired in the processor models ("the language is oriented...at the millisecond level and above")[3]. Procurement and training costs might be a problem because the language is so relatively new. However, the Army, in conjunction with FEDSIM, has recently used ECSS-II for simulating a communication system with favorable results[5].

In conclusion, a simulation language such as GPSS or SIMSCRIPT II would be more versatile for the LDCN modeling effort than FORTRAN. However, variations in the software design of the processors would be more easily evaluated in FORTRAN because of its extended I/O capabilities. GPSS is probably too general for this effort, however, it does have some merits. If SIMSCRIPT II were to be selected as the language to use, serious consideration should be given to the ECSS-II language, as it is based on SIMSCRIPT II, while providing a closer fit to the LDCN application.

9. CONCLUSIONS

The LDCN is a complex system, composed of major subsystems in concentrators, Front End Processors, and Hosts, unified via data communications lines into an operational system. Major design issues can focus on the network as a whole, or on any subsystem, and may deal with hardware or software. Development of a simulation model as a general tool to enable examination of such issues requires major considerations be given first to the overall modeling stragety. The strategy must address the issues of modeling the individual subsystems as well as modeling their interaction in a unified system. The purpose of this report has been to present an appropriate strategy for modeling the LDCN.

The major conclusions of this report are as follows:

- Because of the size of LDCN, one large program that allows detailed processor modeling as well as global network modeling is infeasible.

- An appropriate strategy is hierachically structured; at the highest level is a similified global network model with the processors modeled as simple single-servers. This model serves two purposes;

    1) It will serve as the framework for the detailed processor models as they are built;

    2) It will provide the first gross performance trends quickly and cheaply as it evolves into an accurate global performance measurement tool.

- The detailed processor models are extracted from the global model by replacing one simple server model of a processor with the detailed version of the model, removing all servers of the same class, removing all servers that are not directly connected via communications lines to the detailed servers and finally reduce the scope of the model until it becomes tractable.

9.1

- When modeling communications processors, the external environment is very important. Simplifications and omissions must be treated carefully to prevent inaccurate results.

- The element that links the detailed processor models together is the service time distribution. Initially, best guess estimations are used to characterize the service time of the external processors. Measurements are obtained from the detailed model and these distributions are then used to model the environment of a different processor.

- The service time distributions are used in the global network model to obtain more accurate global performance statistics.

- The common simulation program structure is composed of a monitor, utilities and the models themselves. The utilities are made up of a Front-End for input and output, a traffic generator and a statistics package.

- The detailed portion of the processor models can be structured in two ways:

    1) Inbed the software design of the processor directly into the simulation code. This requires no program input, but the evaluation of variations in the software design requires repeated simulation program modification.

    2) The software design is read in as input, consisting of cycle vectors - the sequence of servers or facilities visited by transactions, and state vectors - the state induced on each server in the cycle vector. A skeleton program is designed that reads in this input to create different "programs."

- The input to the simulation programs consists of:

9.2

1) System (network or processor) configuration including the linkage information;

2) Software design - if implemented with the vector approach;

3) Demand and service characteristics.

- The output from the simulation programs includes:

1) Statistics describing system (network or processor) behavior;

2) Linkage information to be used in other programs.

- Because of the excessive amount of interactions between the various programs, the order in which the programs are run is important. Care should be taken to avoid repetitions of runs and inaccurate results.

- Three languages that could be used to write the simulation programs are FORTRAN, GPSS, and SIMSCRIPT II. Each has its own merits and drawbacks. Of the three, SIMSCRIPT II would probably be the best, with GPSS and FORTRAN tied.

- However, new languages are available that are designed specifically for simulating computer system. One such example is ECSS-II, written at RAND. Based on SIMSCRIPT II, ECSS-II has built-in features that relate directly to computer hardware and software.

REFERENCES

1. Reitman, J., <u>Computer Simulation Applications</u>, Wiley-Interscience New York, 1971.

2. Kosy, Donald, <u>The ECSS II Language for Simulating Computer Systems</u>, The RAND Corporation, Santa Monica, CA, December, 1975.

3. Kosy, Donald, "An Interim Empirical Evaluation Evaluation of ECSS for Computer System Simulation Development," <u>Proceedings of the Symposium on the Simulation of Computer Systems</u>, National Bureau of Standards, Gaithersburg, MD, June, 1973.

4. Feingold, R. and Y. Chao, "Statistical Instrumentation of ECSS Models," <u>Proceedings of the Second Symposium on the Simulation of Computer Systems</u>, National Bureau of Standards, Gaithersburg, MD, June, 1974.

5. Sprung, J., P. Beatty, J. Camery, and J. Norrell, "An Introduction to the Simulation of a Multiple CPU Military Communications System," <u>Proceedings of the Fourth Symposium on the Simulation of Computer Systems</u>, National Bureau of Standards, Boulder, CO, August, 1976.

BIBLIOGRAPHY

1. Chou, W. and P. McGregor, "A Unified Simulation Model for Communication Processors," Proceedings of Trends and Applications 1975: Computer Networks, Symposium at NBS, June 1975.

2. McGregor, P., W. Chou, and R. Kaczmarek, "Communications Processor Simulation: A Practical Approach," Proceedings of the National Telecommunications Conference, December 1975.

3. Chou, W., H. Frank, and R. Van Slyke, "Simulation of Centralized Computer Communications Systems," Proceedings of 3rd Data Communications Symposium: Data Networks Analysis and Design, November 1973.

4. Gordon, G., System Simulation, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1969.

5. Martin, F., Computer Modeling and Simulation, John Wiley & Sons, New York, 1968.

6. Van Slyke, R., W. Chou, and H. Frank, "Avoiding Simulation in Simulating Computer Communication Network," Proceedings of the National Computer Conference, Montvale, NJ, Vol. 42, AFIPS Press, 1973, pp. 165-167.

7. Maisel, H. and G. Gnugnoli, Simulation of Discrete Stochastic Systems, Science Research Associates, Inc., Chicago, 1972.

8. Network Analysis Corporation, "Capacity Analysis of Data Communications Concentrator," April 1975.

9. Network Analysis Corporation, "Communications Processor Simulation Program (CPSP) - Users Manual," May 1975.

10. Network Analysis Corporation, "Feasibility Analysis of PDP-11 Mini-computer Front-End Processor," January 1976.

Network Analysis Corporation

DISTRIBUTION LIST:

NAC:

    J. Eckl

    R. Kaczmarek

    P. McGregor

    Project 95 File (Accounting Office)

Other:

    J. Sowa

    J. Alexander

    G. Mountz

    Code 9642, FMSO

    Mechanicsburg, PA   17055

    Richard des Jardins

    Code 510

    NASA/Goddard Space Flight Center

    Greenbelt, MD   20771